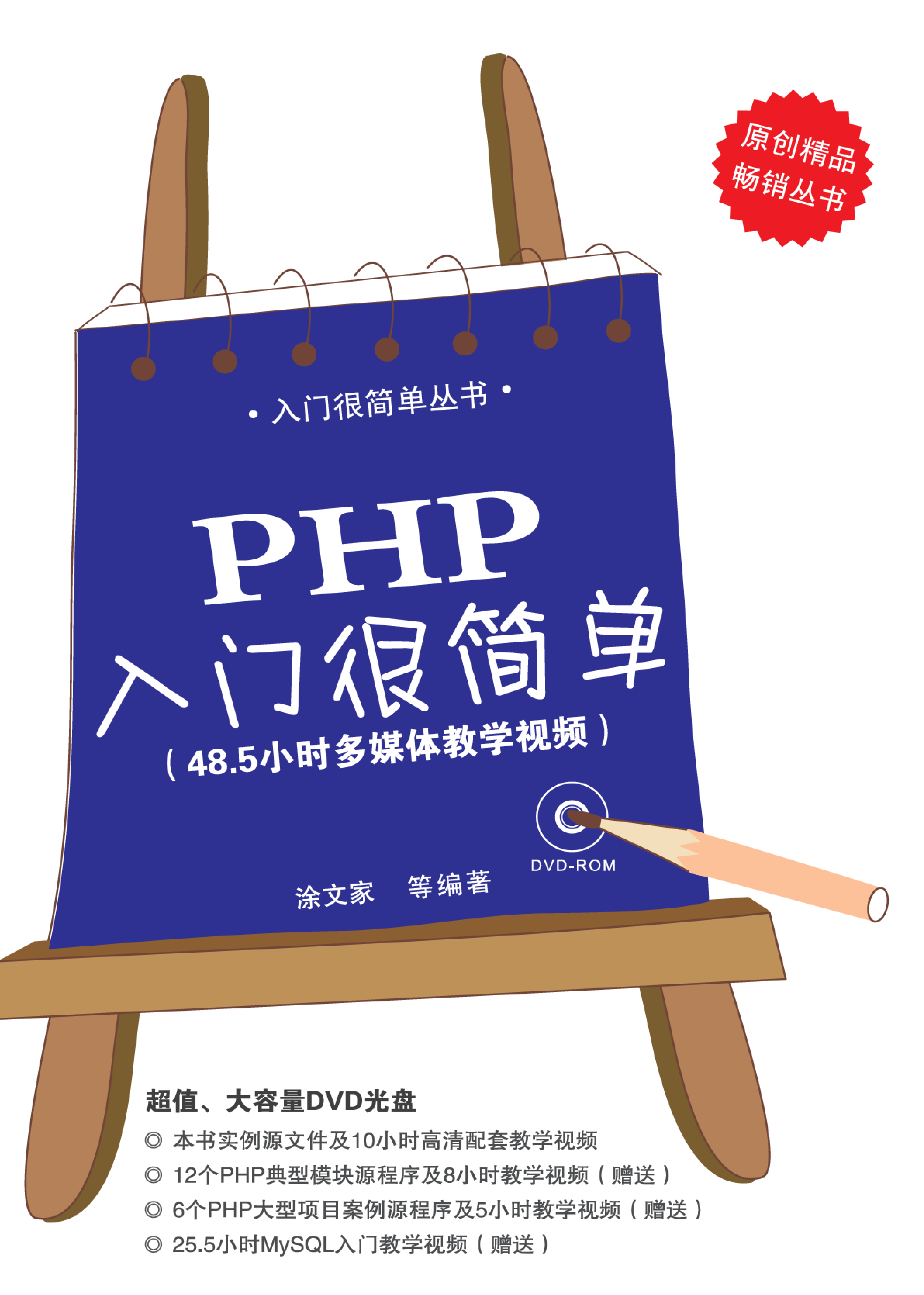
浅显易懂，零门槛学习PHP脚本编程，真的很简单

娓娓道来，亲切自然，老鸟带领菜鸟，快速跨入PHP的大门

161个典型实例、9个案例、43个习题，通过动手实践提高开发水平



超值、大容量DVD光盘

◎ 本书实例源文件及10小时高清配套教学视频

◎ 12个PHP典型模块源程序及8小时教学视频（赠送）

◎ 6个PHP大型项目案例源程序及5小时教学视频（赠送）

◎ 25.5小时MySQL入门教学视频（赠送）

清华大学出版社

入门很简单丛书

PHP 入门很简单

涂文家　等编著

清华大学出版社
北　京

内 容 简 介

本书以简单、轻松的语言细致地介绍了 PHP 开发的相关知识。书中的每章内容都是 PHP 开发的重点。本书讲解由浅入深，通过大量实例和详细的代码及代码注释让读者理解和掌握相应的知识点，并提供了大量习题供读者演练，以检测和巩固学习效果。**另外，作者专门为本书录制了配套多媒体教学视频，以辅助读者高效、直观地学习。这些视频及本书涉及的源代码一起收录于配书光盘中。**

本书共 14 章，分为 5 篇。第 1 篇为初识 PHP 脚本语言，介绍了 PHP 的基本定义、特点、原理及 PHP 文件等；第 2 篇介绍了 PHP 中的常量、变量和数据等相关知识点；第 3 篇为 PHP 编程基础，介绍了条件与循环语句、脚本的重用、Web 编程基础及数据的存储等；第 4 篇为面向对象编程，介绍了 PHP 与操作系统、PHP 与基于对象的编程（OOP）、PHP 与 MVC 等；第 5 篇为开源 PHP 应用，主要介绍了 WordPress 和 Drupal 两个常见开源 PHP 应用。

本书着重夯实基础，基本覆盖了 PHP 开发的基础知识，特别适合 PHP 初学者打好基本功这个阶段时阅读，也适合有一定开发经验的读者查阅和参考。另外，本书还适合大中专院校作为相关专业的教材。

图书在版编目（CIP）数据

PHP 入门很简单 / 涂文家等编著. —北京：清华大学出版社，2014
（入门很简单丛书）
ISBN 978-7-302-35564-9

Ⅰ. ①P…　Ⅱ. ①涂…　Ⅲ. ①PHP 语言－程序设计　Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字（2014）第 038501 号

责任编辑： 夏兆彦
封面设计： 欧振旭
责任校对： 徐俊伟
责任印制：

出版发行： 清华大学出版社
网　　址： http://www.tup.com.cn, http://www.wqbook.com
地　　址： 北京清华大学学研大厦 A 座　　**邮　　编：** 100084
社 总 机： 010-62770175　　**邮　　购：** 010-62786544
投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销： 全国新华书店
开　　本： 185mm×260mm　**印　　张：** 25.75　**字　　数：** 646 千字
（附光盘 1 张）
版　　次： 2014 年 8 月第 1 版　　**印　　次：** 2014 年 8 月第 1 次印刷
印　　数： 1～ 000
定　　价： 　　元

产品编号：054560-01

前　　言

PHP（Hypertext Preprocessor，超文本预处理语言）承担的任务是在用户向 Web 服务器发出超文本请求之后，在 Web 服务器向用户返回被请求的页面之前，由 PHP 引擎来对用户请求的页面进行预处理。它是一种 HTML 内嵌式的语言，可以很方便地被嵌入到 HTML 页面中。这一点与 ASP 类似。但是，PHP 的开源特性、跨平台特性、小巧的体积和高效的运行效率，以及开源社区的广泛支持和它支持众多的数据库等特性却是它被广泛使用的根本原因。

1. 开源、免费

PHP 发布于 1995 年。从那之后，发展非常迅速。这都归功于它的开源特性。在开源协议的框架下，用户甚至不需任何花费，就可以获得各种升级和安全补丁，保证 PHP 引擎的安全。

2. 广泛支持

PHP 的快速发展也离不开大家的支持。在互联网上，人们讨论最多的编程语言就是 PHP。如果你在使用过程中发现了任何解决不了的问题，打开搜索引擎，输入问题的关键词，一定可以找到解决方案的。

3. 支持众多的数据库

PHP 对数据库的支持也是其深受欢迎的原因之一。PHP 支持通过 ODBC 连接各种类型的数据库，同时也针对 MySQL 数据库开发了便捷的连接和操作方式。由于其对 MySQL 数据库的原生支持，使 MySQL 数据库成为了最受欢迎的开源数据库软件。

本书特色

1. 诙谐幽默，接地气儿

本书试图以生动活泼的语言、接地气儿的讲解方式来为读者呈现 PHP 的世界。本书讲解知识点时和日常生活中的方方面面联系起来，让读者更加容易理解抽象的编程世界。作者希望通过这种接地气儿的语言和内容组织形式，让读者在程序员的道路上不至于越走越累，从而充分领略到 PHP 的魅力。

2. 夯实基础，注重实用

本书涵盖读者初涉 PHP 开发所需要掌握的各种基础知识。通过本书，可以为读者后续

的 PHP 应用开发夯实基础。另外，本书讲解知识点时列举了大量生动有趣的实例，可以大大提高读者的实际编程能力。

3．代码精讲，注释丰富

本书中的代码大部分给出了详细的解释，并且进行了丰富的注释，便于读者阅读和理解，也可以培养读者从一开始就养成良好的编码习惯。

4．实例真实，生动有趣

就像作家写小说需要从实际生活中汲取生活经验一样，程序员在编写程序的过程中也需要从实际生活经验中找灵感。本书中的实例大多来源于日常生活。这些例子生动有趣，可以提升读者对日常生活的观察力，从而在日常生活中找到编程的灵感。

5．视频教学，高效直观

作者专门为本书录制了配套多媒体教学视频，以帮助读者更加直观、高效地阅读本书，达到事半功倍的学习效果。

6．实战练习，巩固提高

本书除了第 14 章之外，其他各章最后都提供了练习题，可以帮助读者巩固和提高所学的知识，也可以方便相关老师教学时使用。

本书内容及体系结构

第 1 篇　初识 PHP 脚本语言（第 1~3 章）

本篇介绍了 PHP 脚本的工作原理，并通过在 Windows 平台上搭建 PHP 开发环境进一步加深读者对 PHP 脚本语言工作原理的认识。

第 2 篇　常量、变量与数组（第 4~6 章）

本篇介绍了 PHP 脚本中经常使用的常量和变量，以及作为变量集合的数组。在这三章中，读者可以了解这些概念背后的意义及何时需要使用它们。本篇通过大量的示例和有的放矢的实战练习，帮助读者更好地掌握这些知识。

第 3 篇　PHP 编程基础（第 7~10 章）

本篇介绍了 PHP 中的执行流程和控制机制。通过这些控制机制，可以实现在指定条件下运行相应的脚本、循环使用脚本及脚本的大规模复用。有了这些流程控制机制，程序员便可以用较少的代码来完成各种纷繁复杂的任务，提高开发效率。另外，本篇还介绍了 Web 编程的基础和数据存储的相关知识，为使用 PHP 进行 Web 开发打好基础。

第 4 篇　面向对象编程（第 11~13 章）

本篇主要介绍了 PHP 与操作系统、PHP 与基于对象的编程（OOP）及 PHP 与 MVC

框架等内容。通过这三章内容的学习，读者可以更加深刻地认识 PHP 面向对象编程的特性。

第 5 篇　开源 PHP 应用（第 14 章）

本篇主要介绍了 WordPress 和 Drupal 两款知名的 PHP 应用。读者在掌握了前面各章内容后，可以使用这两款 PHP 应用来快速搭建自己的网站。

本书超值 DVD 光盘内容

- 本书各章涉及的实例源文件；
- 10 小时本书配套教学视频；
- 12 个 PHP 典型模块源程序及 8 小时教学视频；
- 6 个 PHP 大型项目案例源程序及 5 小时教学视频；
- 25.5 小时 MySQL 入门教学视频。

本书读者对象

- 没有任何基础的 PHP 初学者；
- PHP 开发的爱好者；
- 刚入职的 PHP 初中级程序员；
- 大中专院校的师生；
- 相关培训学校的学员。

本书作者

本书由涂文家主笔编写，其他参与编写的人员有丁士锋、胡可、姜永艳、靳鲲鹏、孔峰、马林、明廷堂、牛艳霞、孙泽军、王丽、吴绍兴、杨宇、游梁、张建林、张起栋、张喆、郑伟、郑玉晖、朱雪琴、戴思齐、丁毓峰。

虽然笔者花费了大量精力写作本书，并力图将疏漏减少到最少，但仍恐百密一疏。如果您在阅读本书的过程中发现有任何疏漏，或者对本书讲解有任何疑问，都可以与作者取得联系。E-mail 地址：bookservice2008@163.com。

编著者

目　　录

第 1 篇　初识 PHP 脚本语言

第 2 篇 常量、变量与数据

第 3 篇 PHP 编程基础

第 4 篇 面向对象编程

第 5 篇 开源 PHP 应用

第 1 篇　初识 PHP 脚本语言

⏭　第 1 章　什么是 PHP

⏭　第 2 章　搭建 PHP 运行环境

⏭　第 3 章　动手写第一个 PHP 脚本

第 1 章　什么是 PHP

当你拿起这本书的时候，起码应该听说过 PHP。无论你是否明白这三个字母分别代表哪几个单词，起码应该对 PHP 有一些了解，并想深入学习这门脚本语言。本章将围绕 PHP 的定义、为什么要使用 PHP 及 PHP 是如何工作的这几个问题来展开讨论。

1.1　PHP 的定义

按照 PHP 官方网站首页（http://www.php.net）给出的定义，PHP 是一种通用的脚本语言。因其特别适合 Web 开发，并且可以嵌入 HTML 语言而得到广泛的使用。维基百科则指出，PHP 是最早出现的几种可以嵌入 HTML 源文件的服务器端脚本语言之一。安装了 PHP 处理模块的 Web 服务器可以通过解析 PHP 代码来动态地生成最终的页面。简而言之，PHP 是一种开发动态网页的通用服务器端脚本语言。

在这一段定义 PHP 的文字中，有这样几个关键词：动态网页、通用编程语言和服务器端脚本语言。在本节中，将围绕这三个概念及与之相对应的一些概念来了解什么是 PHP。

1.1.1　动态网页 vs. 静态网页

根据上文的定义，PHP 是一种用于开发动态网页的脚本语言。那么，什么是动态网页呢？如果存在动态网页，那么肯定有一种网页叫静态网页，那么，什么又是静态网页呢？本小节主要讨论这两个问题。

1．动态网页

维基百科将动态网页定义为：在用户访问时或与用户交互时实时生成或发生改变的网页。这一概念是 Web 2.0 时代的基础概念。正是有了动态网页这个概念，跨站信息共享才成为可能。为了更深入地了解动态网页的运作机制，可以参考图 1-1。



图 1-1　动态网页工作原理

在图 1-1 中，通过“本地计算机”访问到的网页如果是动态网页的话，那么一般是在“本地计算机”向“服务器”发送请求后，由“服务器”从服务器存储中获取实时数据并嵌入用户请求的页面中，然后返回结果给“本地计算机”。

2. 静态网页

与动态网页会在用户发起访问时实时生成网页不同，静态网页所呈现的内容与存储在服务器上的内容是一模一样的，不会根据访问者发起访问的时间、地点或者其他因素的不同而发生变化。静态网页一般以 HTML 格式存储在服务器上并通过 HTTP 协议呈现给网页浏览者。

1.1.2 通用编程语言 vs. 基于域的编程语言

通用编程语言是指可以用来为不同的应用域（Application Domain）编写应用程序的编程语言。若干个应用域中的应用程序即使是使用同一种通用编程语言编写的，这些应用程序也不会相互影响。究其原因，系统会为每一个应用域分配一个独立的虚拟地址空间（Virtual Address Space）。而操作系统就是根据虚拟地址空间来为应用程序分配资源的。

与通用编程语言对应的一个概念就是基于域的编程语言。这一概念与本书的学习无关，故略去。读者如果有兴趣继续探究，请自行查阅相关资料。

1.1.3 服务器端脚本语言 vs. 客户端脚本语言

根据定义可知，PHP 是一种服务器端脚本语言。那么在 B/S 架构下，有服务器端脚本语言，就一定会有客户端脚本语言。本小节将对这两种脚本语言的特点进行比较。

1. 客户端脚本语言

客户端脚本语言是指在一个网页的范围内根据鼠标和键盘的动作或某一时间事件动态改变网页内容的脚本语言。比较常见的客户端脚本语言有 JavaScript 和 ActionScript 等。

2. 服务器端脚本语言

服务器端脚本语言是指服务器根据用户请求做出回应来改变网页的若干内容、调整网页载入顺序或重载页面。比较常见的服务器端脚本语言除了 PHP，还有 ASP、ASP.NET 和 JSP。

通过对动态网页、通用编程语言和服务器端脚本语言这三个概念的理解，我们大致上了解了 PHP 脚本语言的功能与作用，用一句话概括就是：通过部署 PHP 处理模块，Web 服务器在回应用户请求时可以根据用户的需要动态更新呈现给用户的网页。

1.2 为什么要使用 PHP

在了解了 PHP 脚本语言的定义和功能之后，PHP 脚本语言是否是唯一的选择，这一问

题引起了我们的兴趣。既然 PHP 脚本语言并不是唯一的一种开发动态网页的通用的服务器端脚本语言，那么为什么要学习并使用 PHP 呢？换言之，PHP 到底有哪些优点，使得人们趋之若鹜呢？

1.2.1　PHP 脚本的特点

1．响应速度快

正是由于 PHP 脚本语言可以内嵌于 HTML 代码之中，Web 服务器可以在非常短的时间内加载一个 PHP 文件并解析文件中内嵌的 PHP 脚本。

2．免费开源

PHP 是一个免费开源的项目，无论是在 Web 服务器上部署 PHP 引擎，还是使用 PHP 代码编写网站，用户都不需要花费一分钱。天下到底还是有免费的午餐。

3．易用性

PHP 脚本的语法简单明了，即使是对于没有任何编程经验的人来说也算得上是通俗易懂的了。

4．跨平台支持

PHP 引擎支持种类繁多的服务器操作系统，如微软视窗操作系统、开源的 Linux、苹果的 Mac OS 及 Unix 操作系统的各种变体。

5．无处不在的技术支持

PHP 官方网站提供了一份邮件讨论组列表。这些讨论组讨论的内容涵盖了诸如 PHP 概述、PHP 和 Windows、PHP 和数据库之类的话题。读者可以根据自己的需要有选择地加入这些讨论组，以便及时的获取支持。另外，互联网上非官方的 PHP 资源也是异常的丰富，通过搜索引擎，可以很方便地找到学习和使用 PHP 过程中碰到的各种问题的答案。

6．无懈可击的安全脚本

只要用户编写的 PHP 脚本语法合乎规范、运行无报错，访问者是不可能窥见 PHP 代码的。这样一来，脚本的安全性也就得到了保证。

7．强大的可定制功能

PHP 的开源许可允许开发者修改 PHP 软件为己所用。

上面的表述中罗列的特点虽然大致能够反映 PHP 为什么受到追捧，但是我们对 PHP 脚本到底能够做些什么仍然一知半解。为了解决这个问题，下面将分四个小节来讲述 PHP 脚本到底都能做些什么。

1.2.2　PHP 脚本和 Web 应用程序

正如前文所提到的那样，在动态脚本出现之前，网站都是由一个个静态的页面组成的，

访问者通过访问页面上的链接在不同的页面间切换。在那个时候，访问者只能被动地接受网页上的信息，而无法与网站的内容进行交互，访问为他们量身定做的页面，进而实现信息的有效传播。

正是基于这样一种认识，人们开发出种类各异的脚本语言来实现网页的动态化，PHP 脚本就是其中之一。由于 PHP 脚本内嵌在 HTML 代码之中，并在发送给用户之前就已经被安装在 Web 服务器上的 PHP 处理模块处理过了，因此用户从 Web 服务器上接收到的回应页面中的 PHP 脚本已经被替换成了相应的 HTML 代码，从而实现了动态响应用户请求的目的。

网站开发者按照一定的规划运用 PHP 脚本编写的网站因具有动态化、交互性的特点而被称为 Web 应用程序。具体来说，PHP 脚本在 Web 应用程序中主要从事如下几项工作：

- 与 HTML 表单进行交互；
- 与数据库通信；
- 生成安全的网页。

虽然 PHP 脚本有着上述的优势，但是它也不是万能的。因为 PHP 是服务器端脚本语言，它不能和用户的浏览器进行交互，也就无法实现对鼠标动作和用户屏幕分辨率的响应。这样一来，如果仅使用 PHP 脚本，则无法实现一些诸如浮动导航菜单之类的特效。

因此在编写网站时，还要结合使用客户端的脚本语言，如 JavaScript，来提升网站易用性和丰富用户体验。

1.2.3　PHP 脚本和数据库应用

PHP 在与数据库交互方面有着无与伦比的优势，因为它支持几乎所有类型的数据库，甚至有些是读者没有听说过的。你只需要告诉 PHP 引擎需要连接的数据库的名字以及数据库的访问路径，PHP 引擎会帮助连接数据库，将你的指令传送到数据库，然后将数据库的处理结果返回给你。

比较常见的数据库类型有：dBASE、Informix、Ingres、Microsoft SQL Server、mSQL、MySQL、Oracle、PostgreSQL 和 Sybase。

除了支持这些常见的数据库之外，PHP 还支持开放数据库互联（ODBC）。通过开放数据库互联，你甚至可以连接到 PHP 无法原生支持的数据库，比如说 Microsoft Access 和 IBM DB2。

1.2.4　PHP 脚本和文件系统

PHP 脚本还可以和文件系统进行交互。换句话说，PHP 可以访问那些存储在本地和其他计算机上可以通过网络访问的文件夹，并在这些文件夹里创建、修改、删除文件或者改变文件属性。你能够在电脑上进行的各种操作基本上都可以通过 PHP 脚本来实现。

许多网络应用程序都会要求与本地的文件系统进行交互。举个例子，某网络应用程序在文件系统中的 Internet 临时文件夹里创建若干文件用来记录相关信息。当用户在同一次会话中再次请求这些信息时，该网络应用程序会将存储在本地的信息发送给用户，而不是从远程数据库中重新读取数据，从而大大地提高了网站的访问速度。

另外，大多数的系统管理和维护脚本也会频繁地和文件系统进行交互。比如，你可以编写一段 PHP 脚本来备份文件，清理文件夹或者处理一些文本文件。

1.2.5　PHP 脚本和系统命令

PHP 脚本还可以向操作系统发出指令，要求操作系统进行相关的操作。例如，你可以通过一段 PHP 脚本执行某系统命令并获得该命令的屏显信息。Windows 操作系统提供了命令行提示符，通过命令行提示符执行 dir 命令来显示当前路径下的文件和文件夹的相关信息。同样，通过编写一段 PHP 脚本，也可以执行同样的操作。

通过 PHP 脚本执行操作系统命令通常会被用来进行系统管理和维护。例如，你想清理某文件夹中特定类型的文件，那么你可以使用 PHP 脚本发出指令要求操作系统列出该文件夹中的所有文件，然后识别并删除指定类型的文件。

除此之外，通过 PHP 脚本还可以运行操作系统中安装的各种应用程序，无论它们是否是用 PHP 脚本编写的。这样一来，我们可以有的放矢的进行应用程序开发：调用已经存在的功能，开发新的功能，既节约了资源，又提高了效率。

1.3　什么是 PHP 文件

在了解了 PHP 脚本之后，还需要关心一个问题：写好的 PHP 脚本应该以什么形式保存起来呢？保存好的 PHP 文件又是如何工作的呢？阅读本节，你就可以找到答案。

1.3.1　PHP 文件的特征

我们可以把编写好的 PHP 脚本保存在一个扩展名为.php 或.phtml 的文件中。一个 PHP 文件可以包含文本、HTML 标签以及 PHP 脚本。在 Web 服务器上安装了 PHP 处理模块之后，Web 服务器就可以处理 PHP 脚本了。

在 PHP 文件中，一段 PHP 脚本通常是以“**<?php**”开头，以“**?>**”结尾的。每一行 PHP 代码都需要以半角分号结束，如同下面这段代码一样。

【例 1.1】 示例代码：

```
<?php echo "<p>Hello World"; ?>
```

在例 1.1 中，“**<?php**”是脚本的开始标签、“**?>**”是脚本的结束标签，而“echo”是一条 PHP 用来输出文本的命令，它告诉 PHP 解析器原样输出关键字后面引号中的内容。Web 服务器在读取到这段代码后就会输出例 1.2 所示并将其发放到用户的浏览器中。

【例 1.2】 示例代码：

```
<p>Hello World
```

用户在浏览器中查看源代码时，就会看到例 1.2 所示中的代码，而不会看到任何 PHP 语句。这样一来，PHP 脚本对于用户来说就像空气一样，看不见也摸不着，脚本的安全性

得到了极大的保障。

1.3.2　PHP 文件是如何工作的

那么是不是任何一台服务器都可以处理 PHP 文件呢？答案当然是否定的。PHP 和 Web 服务器必须要紧密配合才能完成处理 PHP 文件的任务。通常情况下，Apache 软件基金开发的 Apache HTTP 服务器可以做为 Web 服务器来处理 PHP 文件，而 PHP 与 Apache 服务器的配合也是十分默契的。当然，PHP 服务也可以安装在 Microsoft IIS 服务器上，不过配置起来要麻烦的多，具体的内容会在第 2 章中讲到。

在这一小节里，我们重点来回顾一下上一节中讲到的 PHP 文件的处理过程，进而了解 PHP 文件是如何工作的。

PHP 是一种简单易读的高级脚本语言，人们可以像阅读人类语言一样阅读 PHP 脚本，然而电脑却没有办法读懂这些。如果想要电脑读懂 PHP 脚本，需要 PHP 解析器。通过 PHP 解析器，所有的 PHP 脚本都可以被解析成为电脑能读懂的语言。只有这样，电脑才能执行通过 PHP 脚本下发的各种指令。

当用户通过浏览器向某 Web 服务器发出请求后，Web 服务器根据用户的请求打开请求的某 PHP 文件，并在文件中找寻需要处理的 PHP 脚本。当所有的 PHP 脚本根据用户需求被转换成 HTML 代码后，Web 服务器将处理结果，也就是一份包含用户请求信息的纯粹的 HTML 代码文件，发送到用户浏览器。浏览器再将接收到的文件展示给用户。图 1-2 形象地展示了这一过程。

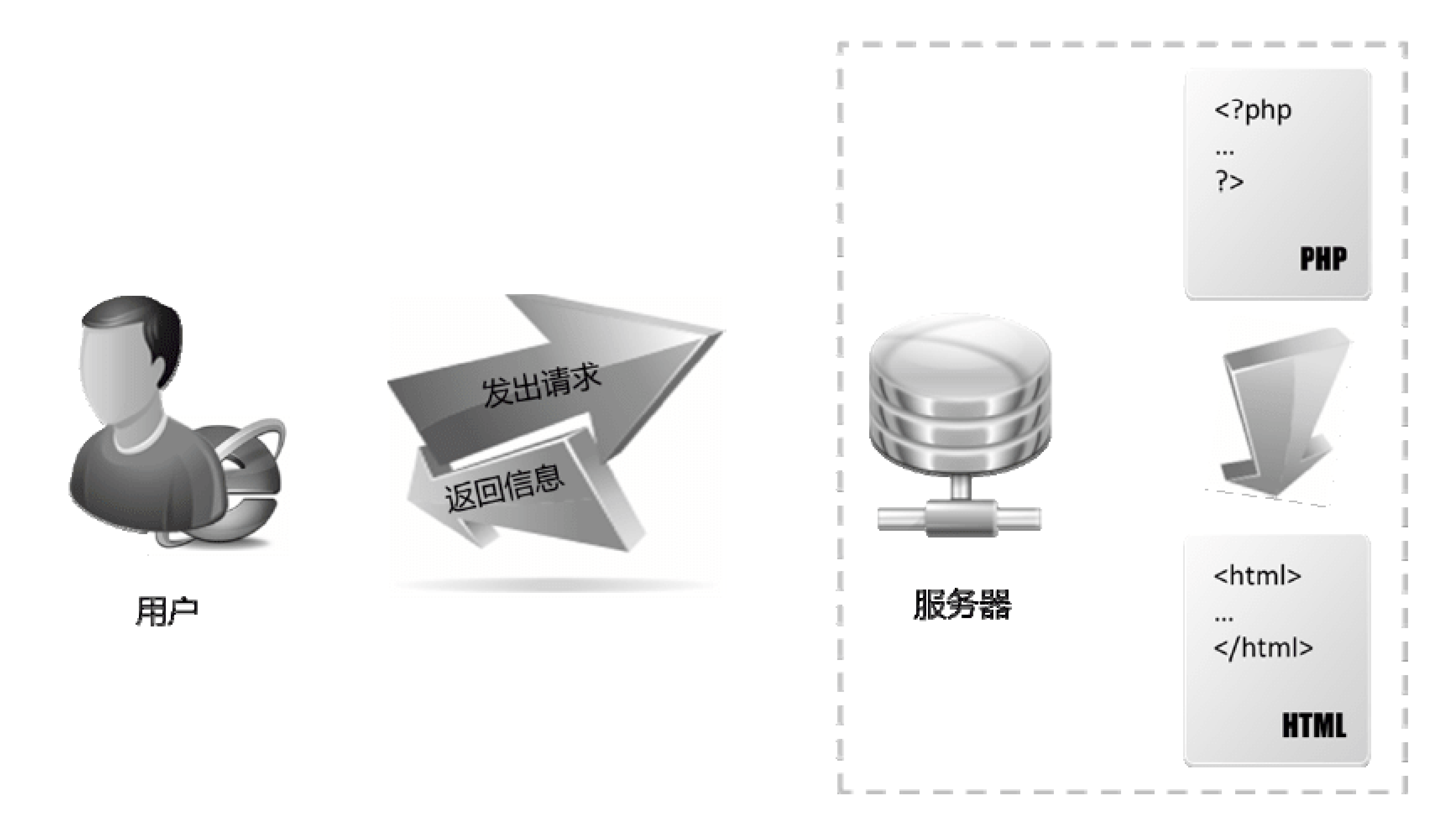


图 1-2　PHP 文件处理机制

1.4　习　　题

（1）什么是动态网页？什么是静态网页？它们有什么区别？

（2）PHP 脚本有哪些特点？

（3）PHP 文件是如何工作的？

第 2 章　搭建 PHP 运行环境

在上一章中，我们了解了 PHP 是一种通用的服务器端脚本语言，这也就意味着 PHP 的运行是需要有服务器支持的。通常情况下，网络上会部署至少两台计算机，一台充当服务器，一台充当客户端。在充当服务器的计算机上安装 PHP 处理模块并上传可供客户端访问的网页。当接收到客户端发出的访问请求时，服务器便使用 PHP 处理模块处理被请求页面中的 PHP 脚本，然后将处理完成的页面返回至客户端的浏览器。

按照这样的说法，岂不是要准备两台电脑？其实不然。为了完成 PHP 运行环境的搭建，一台电脑足矣。

2.1　准备必要的文件

按照 1.3.2 小节中的描述，至少需要一个 Web 服务器软件和 PHP 处理引擎。另外，为了存储网页交互产生的大量数据，还需要搭配一个数据库软件。在本书中，所有的 PHP 页面都是运行在 Apache 服务器、PHP 处理引擎和 MySQL 数据库构建的环境中的。本节将主要讲述如何获取并安装这些组件。

2.1.1　获取 Apache HTTP 服务器软件

Apache HTTP 服务器是由 Apache 软件基金会赞助开发的众多开源软件产品中的杰出代表。读者可以访问 Apache 软件基金会的网站获取更多关于基金会及获得其赞助的软件项目的有关信息。这里需要从官网上下载 Apache HTTP 服务器备用。

访问 Apache 软件基金会的官方网站（http://www.apache.org），单击页面右上角的“项目（Project）”选项。在打开的页面中，单击页面左侧“索引（Index）”项下的“类别（Categories）”子项，进入 Apache 软件基金会项目分类索引，如图 2-1 所示。

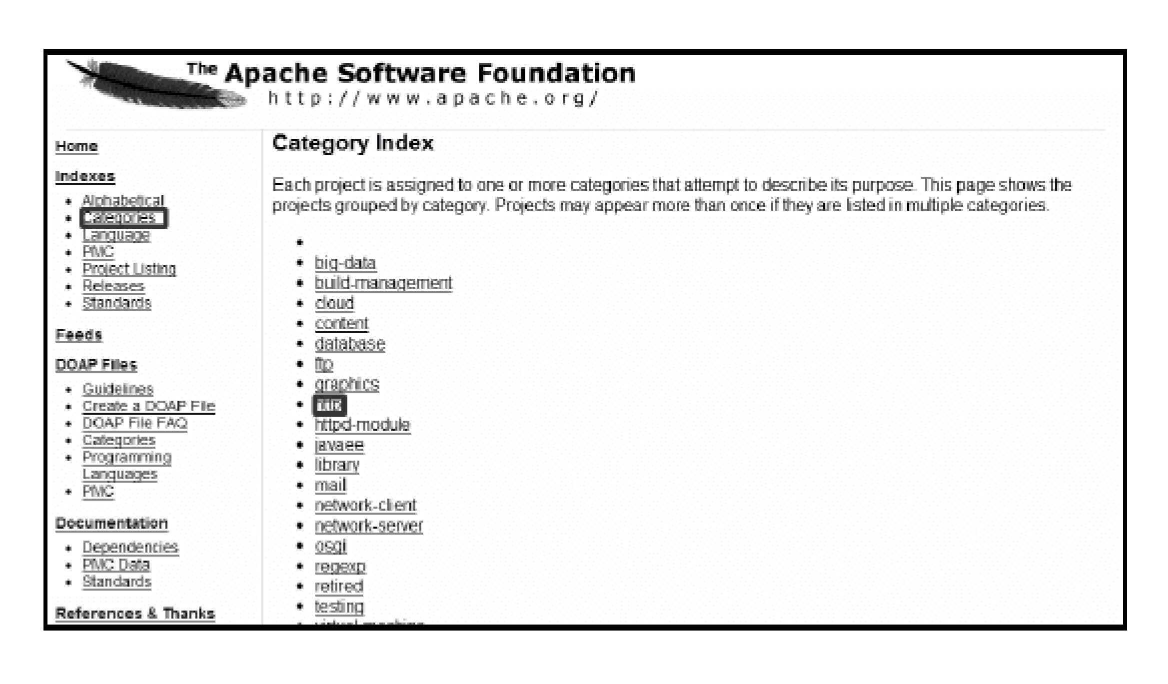


图 2-1　Apache 软件基金会项目分类索引

单击“http”链接后，找到并进入 Apache HTTP Server 项目主页，如图 2-2 所示。

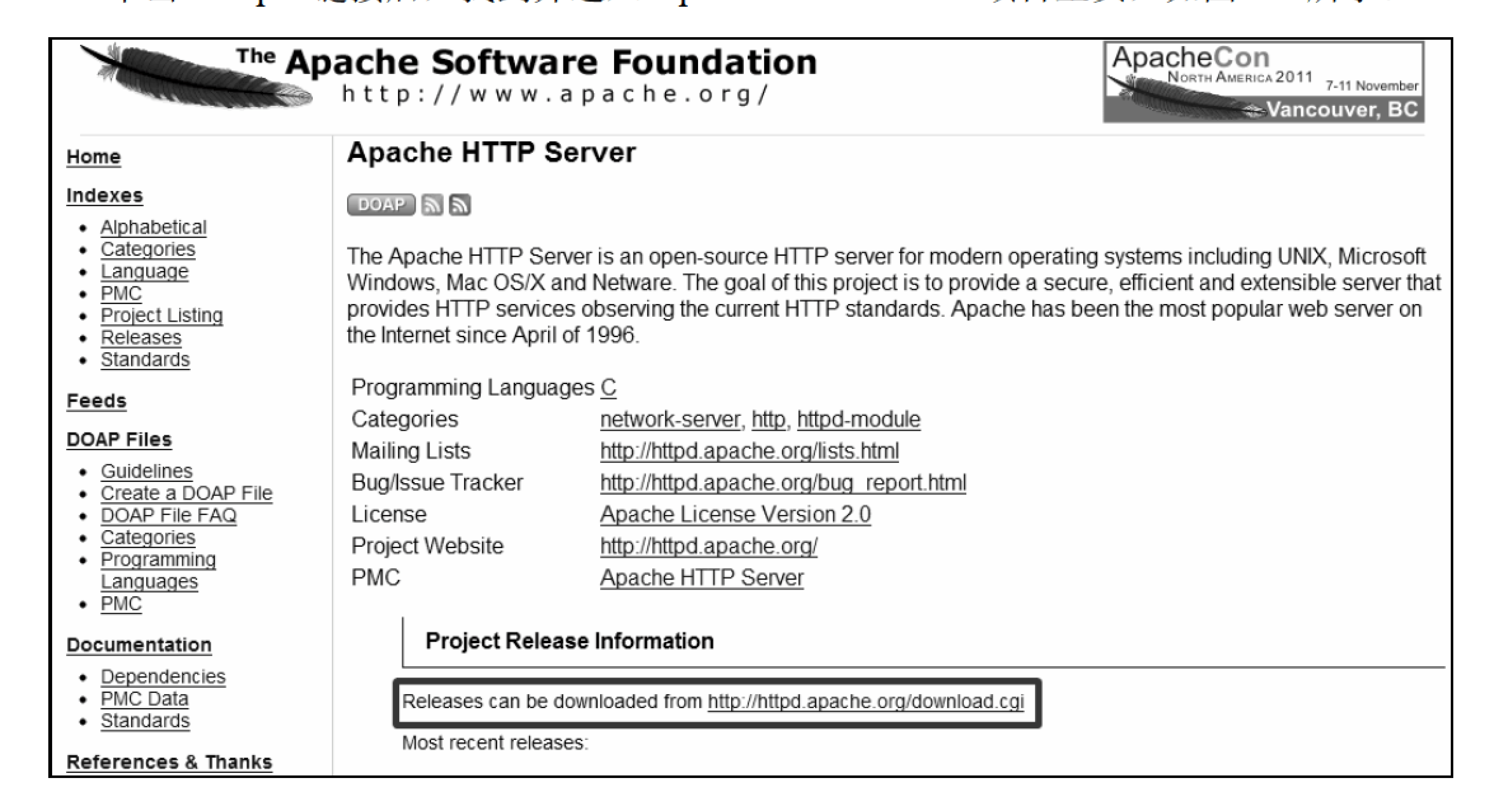


图 2-2　Apache HTTP 服务器项目主页

在主页中找到并单击“由此下载发行版本（Releases can be downloaded from）”的链接进入软件下载页面，如图 2-3 所示。推荐下载最新的稳定版本 2.4.3。

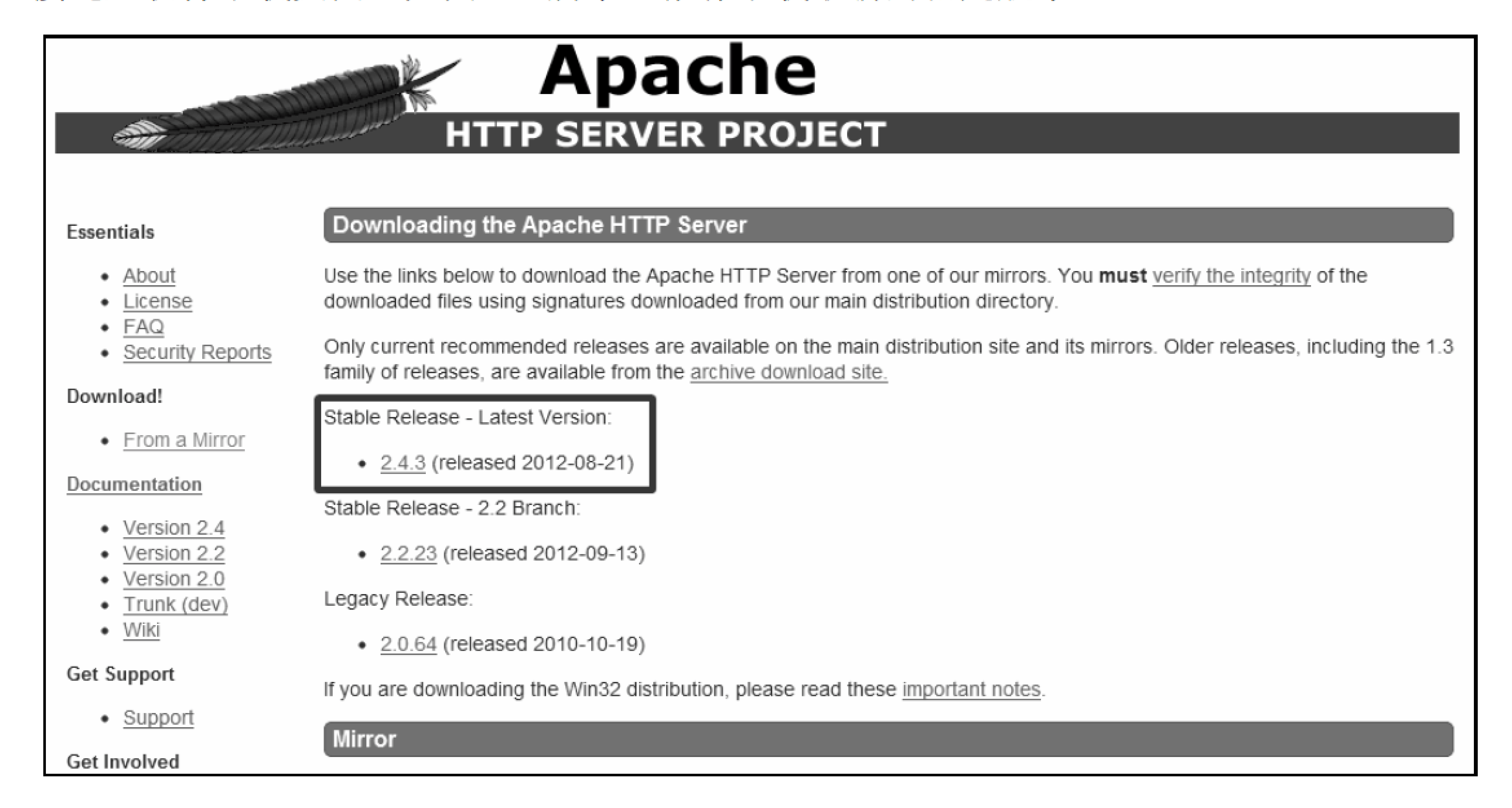


图 2-3　Apache HTTP 服务器下载页面

2.1.2　获取 PHP 处理引擎

PHP 处理引擎通常是两个版本并行开发的，截止 2012 年 10 月 18 日，PHP 的最新版本为 5.4.8 和 5.3.18。官方网站推荐将 PHP 处理引擎升级到这两个版本的其中之一，但是

这两个版本通常被用来和 Microsoft IIS 搭档。因此，本书将使用的 PHP 版本为 5.2.17。

访问 PHP 的官方网站（http://www.php.net），在网站首页上即可找到下载入口，如图 2-4 所示。

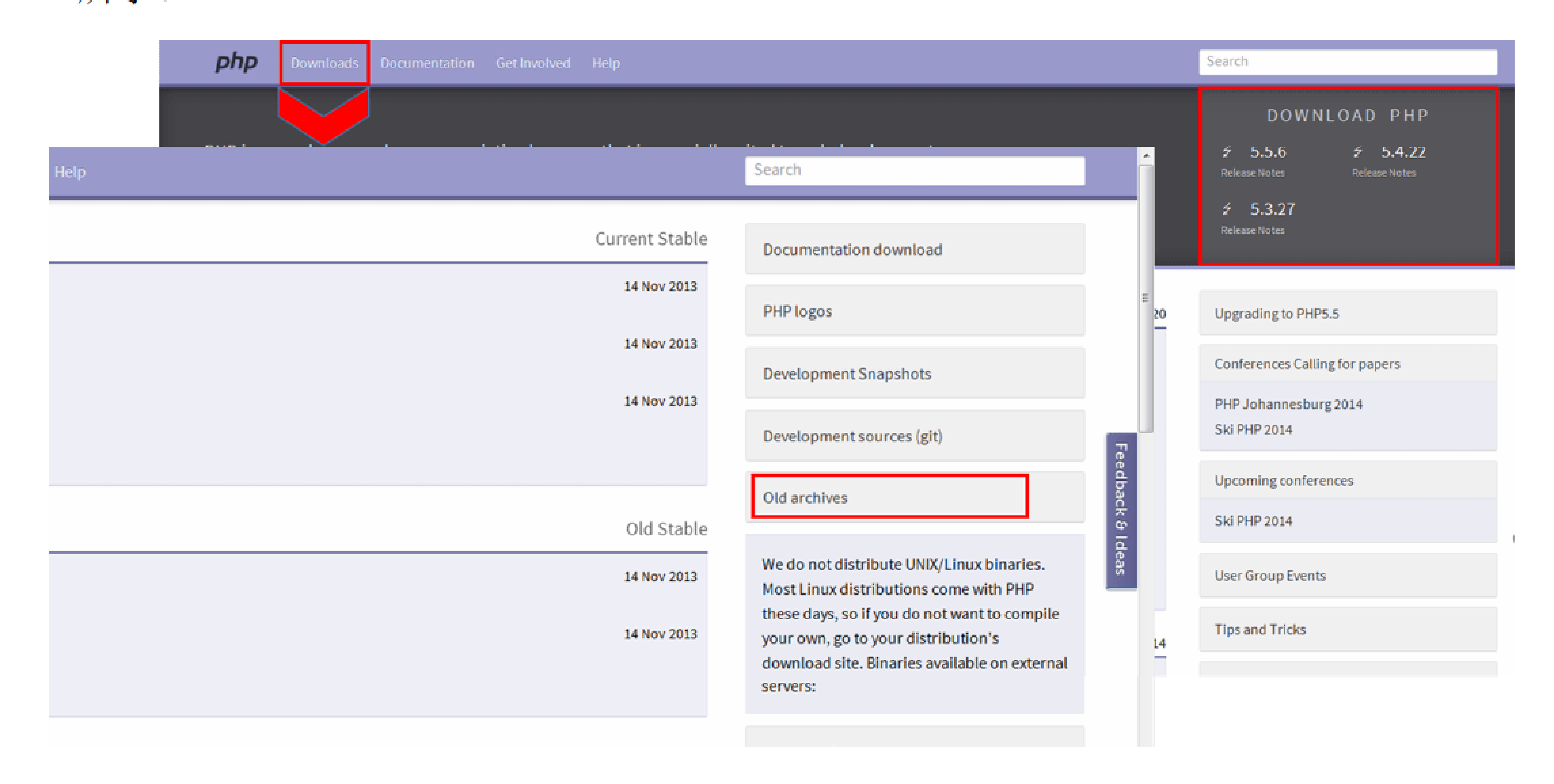


图 2-4　PHP 官方网站首页

在首页的右侧有三个推荐版本的下载链接，同时在导航栏上也有一个下载页面的链接。单击首页顶端导航栏的“Downloads”链接可以进入推荐版本的下载页面；如果需要下载之前的版本，可以单击下载页面右侧的“Old archives”链接进入旧版本的下载页面。本章使用的是 5.2.17 版本的 PHP 引擎，如图 2-5 所示。

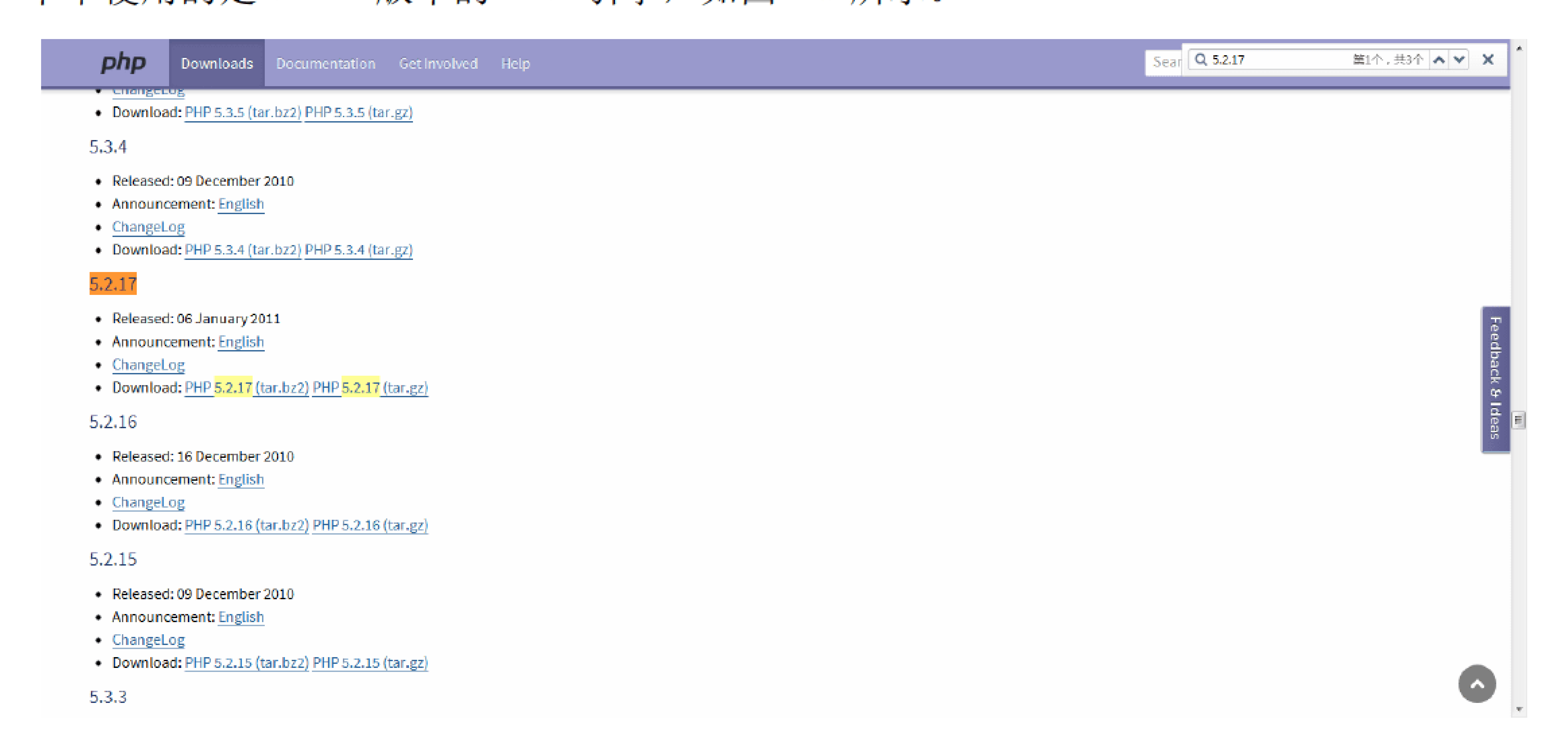


图 2-5　Windows 二进制版 PHP 处理引擎下载页面

2.1.3　获取 MySQL 数据库软件

MySQL 数据库是一款由甲骨文公司（Oracle Inc.）开发的快速、多线程、多用户和健

壮的数据库系统。按照 MySQL 官方网站（http://dev.mysql.com）的说法，MySQL 是全球最受欢迎的开源数据库系统。截止目前，MySQL 的最新版本是 5.2.8。本书将使用最新版本的 MySQL 数据库。

访问 MySQL 官方网站，单击首页上的下载选项卡，进入下载页面。

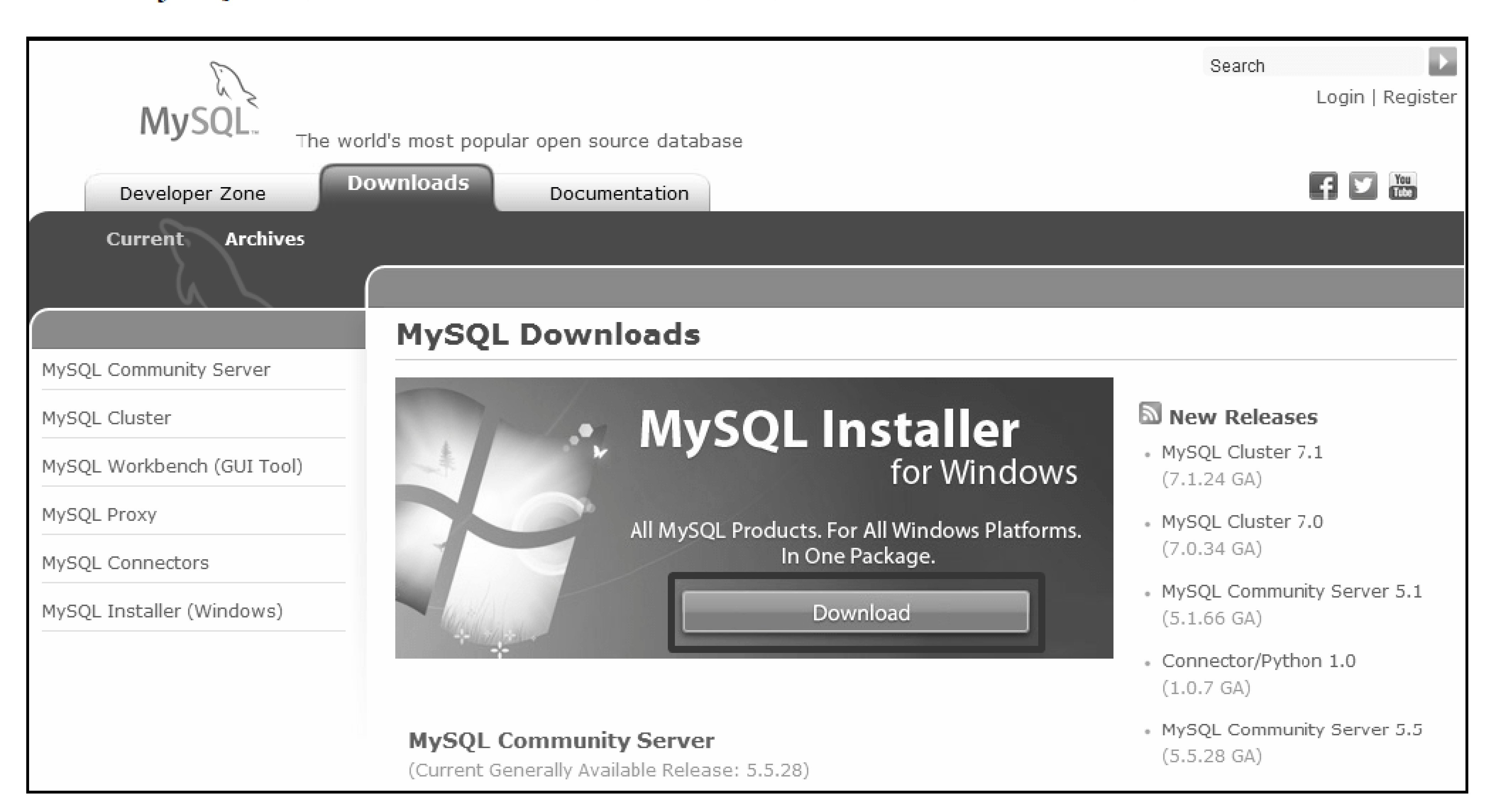


图 2-6　MySQL 数据库下载入口

单击如图 2-6 所示的下载链接，进入 MySQL 数据库安装包的下载页面，如图 2-7 所示。单击“Download”按钮下载安装包。

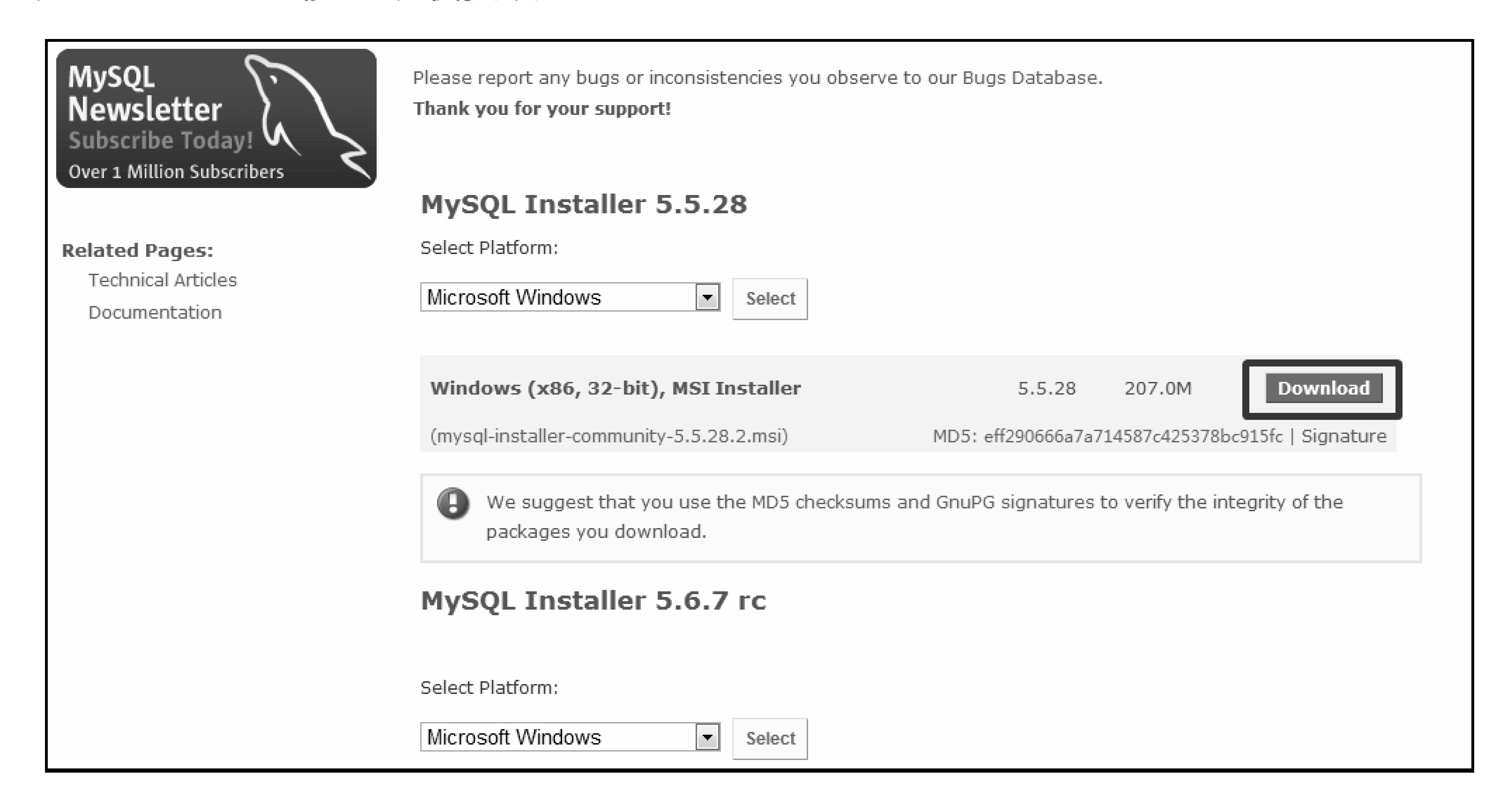


图 2-7　MySQL 安装包下载页面

2.1.4　获取数据库管理软件 phpMyAdmin

phpMyAdmin 是一款完全由 PHP 脚本编写而成的、用于在互联网上管理 MySQL 数据

库的免费软件。它直观的 Web 界面支持对 MySQL 数据的大部分操作。截止目前，phpMyAdmin 的最新版本是 3.5.3，本书将使用 phpMyAdmin 的最新版本。

访问 phpMyAdmin 的官方网站（http://www.phpmyadmin.net）。在页面的右上方，单击如图 2-8 所示的红框内的“.zip”链接转到由 SourceForge 提供的下载页面，如图 2-9 所示。

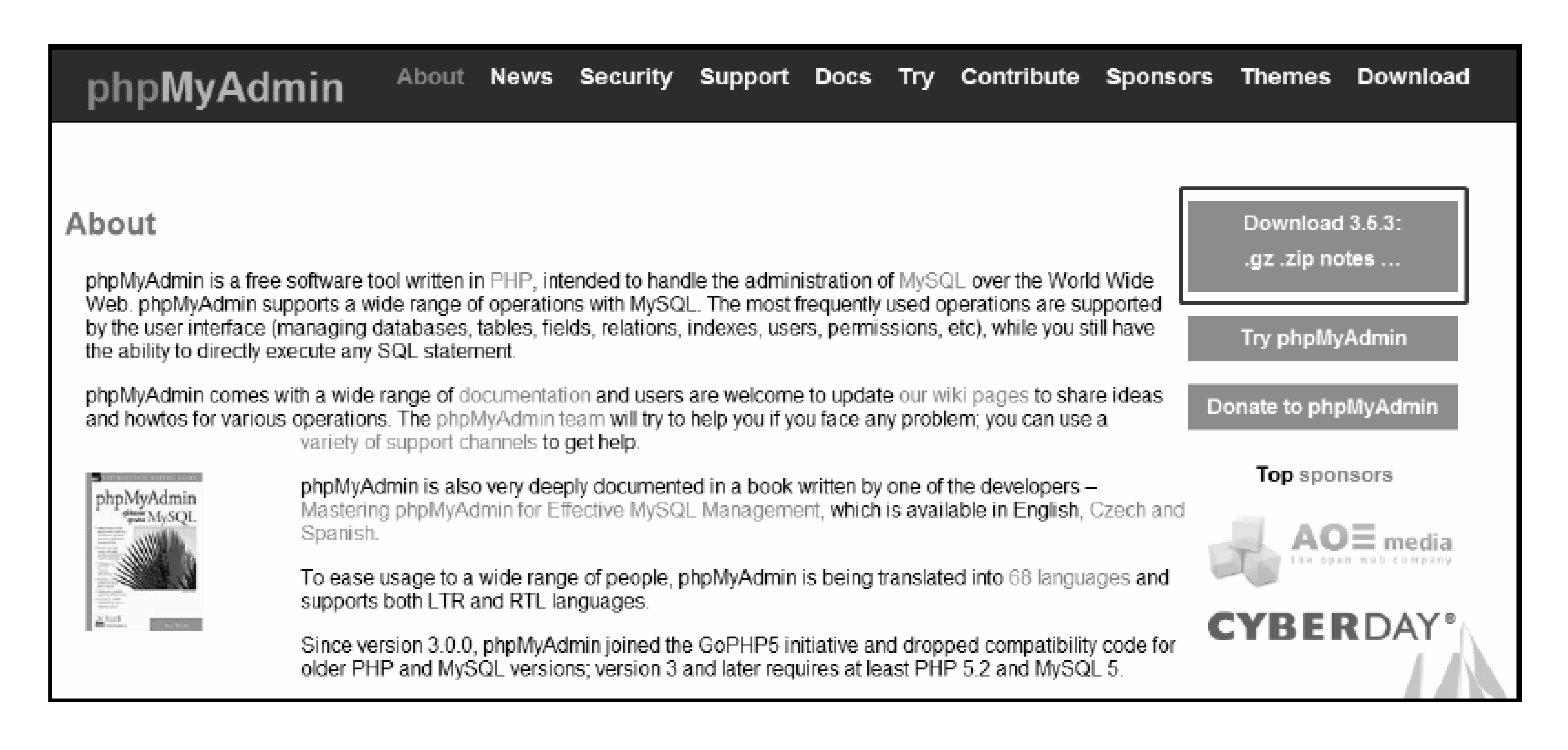


图 2-8　phpMyAdmin 官方网站首页

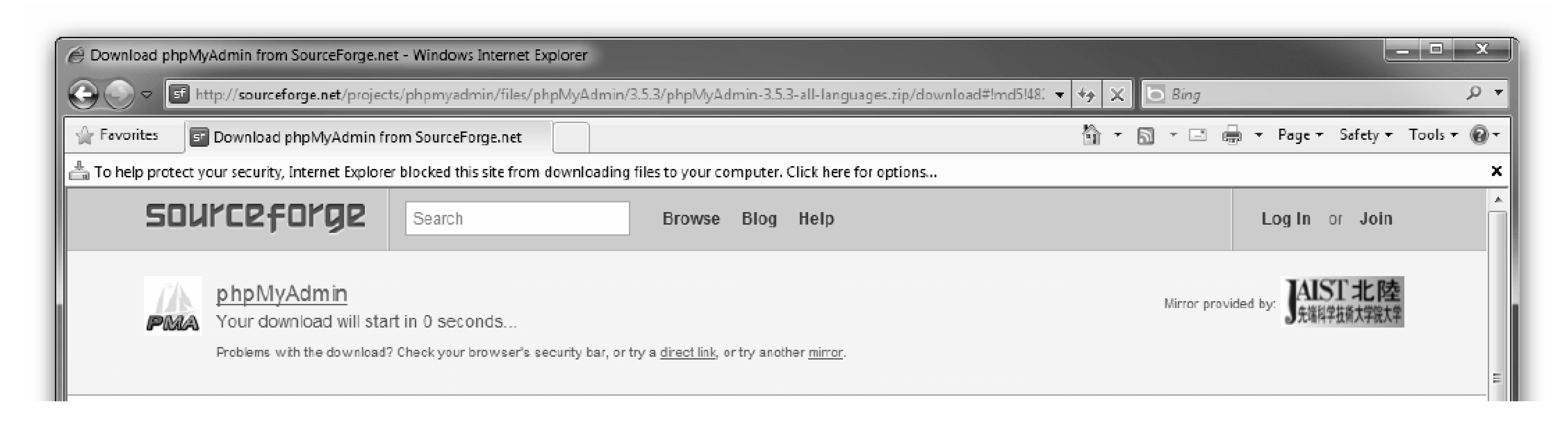


图 2-9　SourceForge 提供的 phpMyAdmin 下载页面

若读者使用的浏览器是 Windows Internet Explorer，浏览器很有可能会出于安全原因阻止文件自动下载，并在页面上方显示提示信息（如图 2-9 中所示的黄色安全提示条）。请在提示信息上单击，并在弹出的菜单中选择“下载文件（Download File）”，页面刷新后，弹出“文件下载（File Download）”对话框，如图 2-10 所示。



图 2-10　文件下载对话框

直接单击“保存（Save）”按钮保存文件到指定目录中备用。

2.2　安装 Apache HTTP 服务器

由于获取的是 Apache HTTP 服务器的 Windows 安装包，因此整个安装过程与其他的 Windows 应用程序类似，操作起来非常的简单。需要注意的是，本书所有的安装示例都是在 Windows 7 英文旗舰版中完成的，如果读者使用的是其他版本的 Microsoft Windows 操作系统，安装过程可能会有些许差别。

2.2.1　安装 Apache HTTP 服务器

当我们双击在上一节中获取的 Apache HTTP 服务器安装包后，Apache HTTP 服务器安装与配置向导开始运行，如图 2-11 所示。单击“下一步（Next）”按钮，可以看到 Apache 许可证授权协议，如图 2-12 所示。



图 2-11　Apache HTTP 服务器安装向导首页

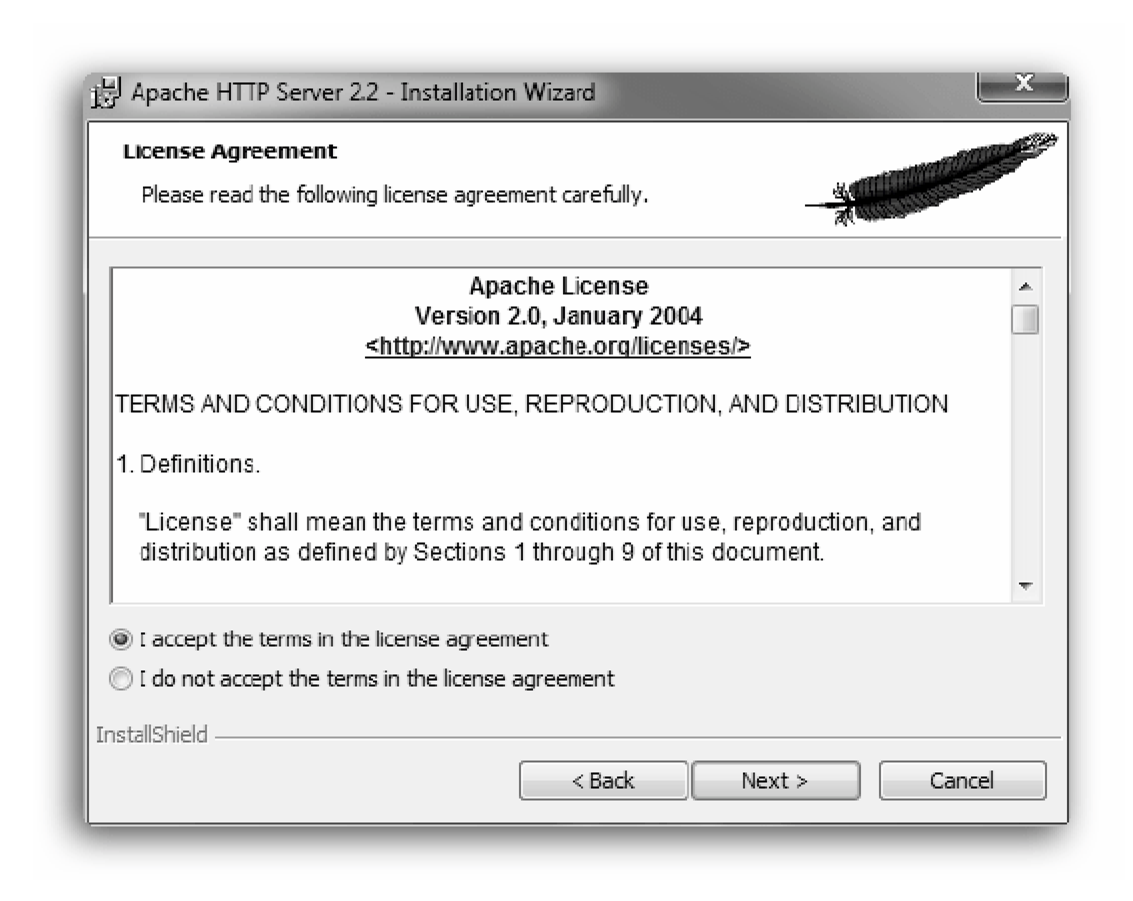


图 2-12　接受 Apache 许可证协议

选择“我接受许可证授权协议中的条款（I accept the terms in the license agreement）”，然后单击“下一步（Next）”按钮，弹出“阅读 Apache HTTP 服务器简介”对话框，如图 2-13 所示。

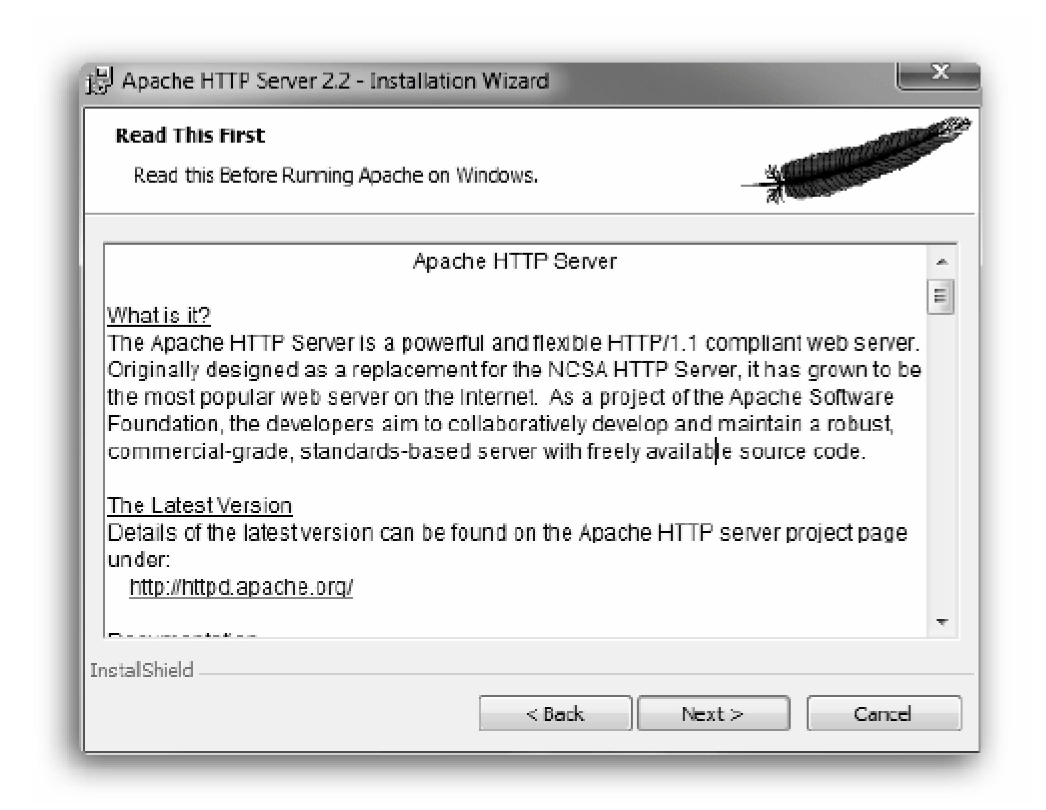


图 2-13　Apache HTTP 服务器简介

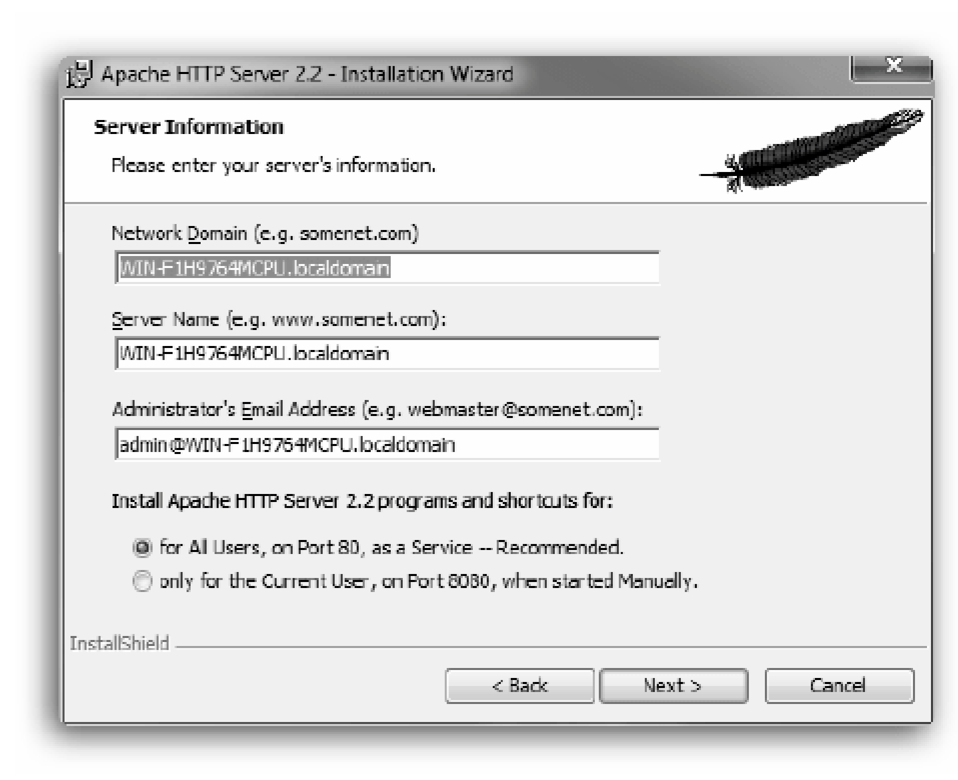


图 2-14　设置服务器信息

在简介中，可以了解 Apache HTTP 服务器是什么及如何获取它的最新版本。阅读完成后，单击“下一步（Next）”按钮，开始设置服务器信息，如图 2-14 所示。

需要设置的有四项，分别是“网络域（Network Domain）”、“服务器名（Server Name）”、“管理员邮件地址（Administrator’s Email Address）”及 Apache HTTP 服务器提供服务的端口号，通常情况下保持默认即可。若当前系统中已经安装了使用 80 端口的应用程序（如 Microsoft IIS），请选择“只为当前用户安装，使用 8080 端口（only for the Current User, on Port 8080，when started Manually.）”。

设置完成后，单击“下一步（Next）”按钮，选择安装类型，如图 2-15 所示。此时，请选择“自定义（Custom）”选项，然后单击“下一步”（Next）按钮。

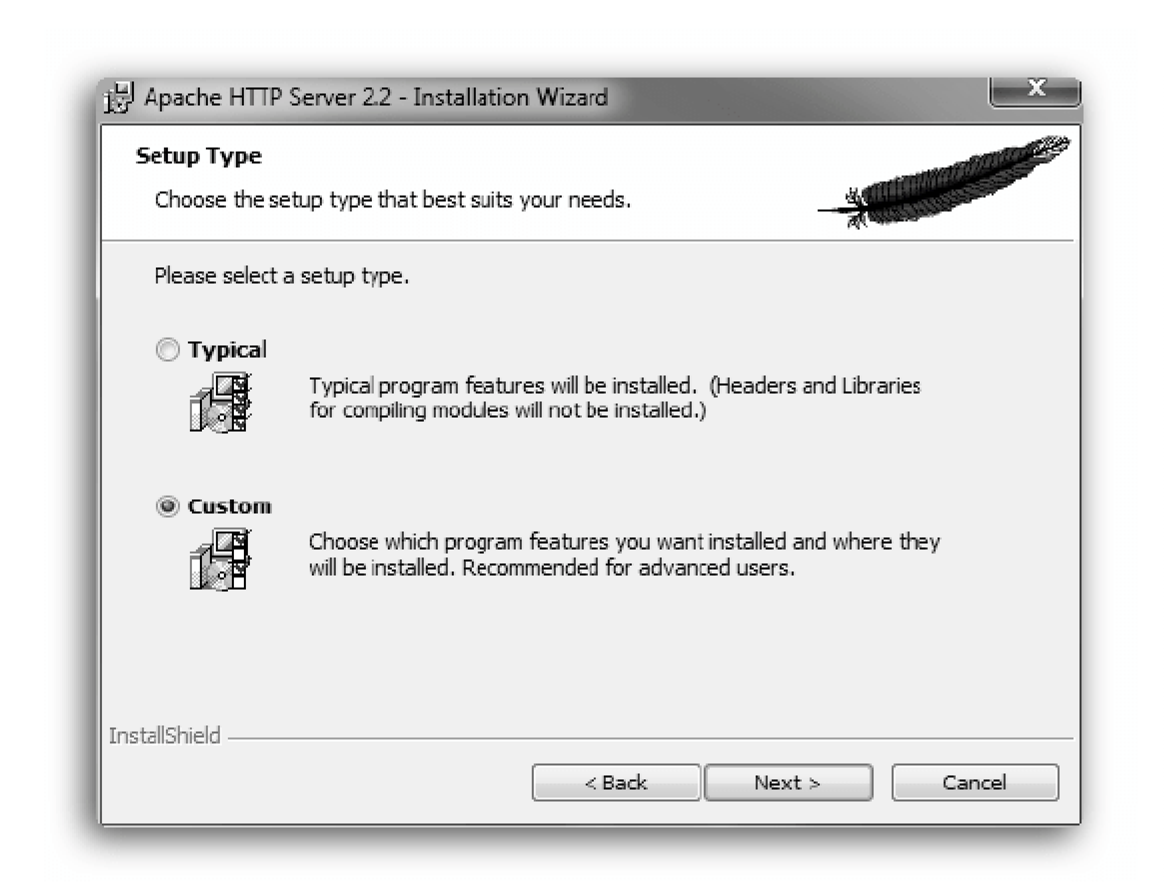


图 2-15　确定安装类型

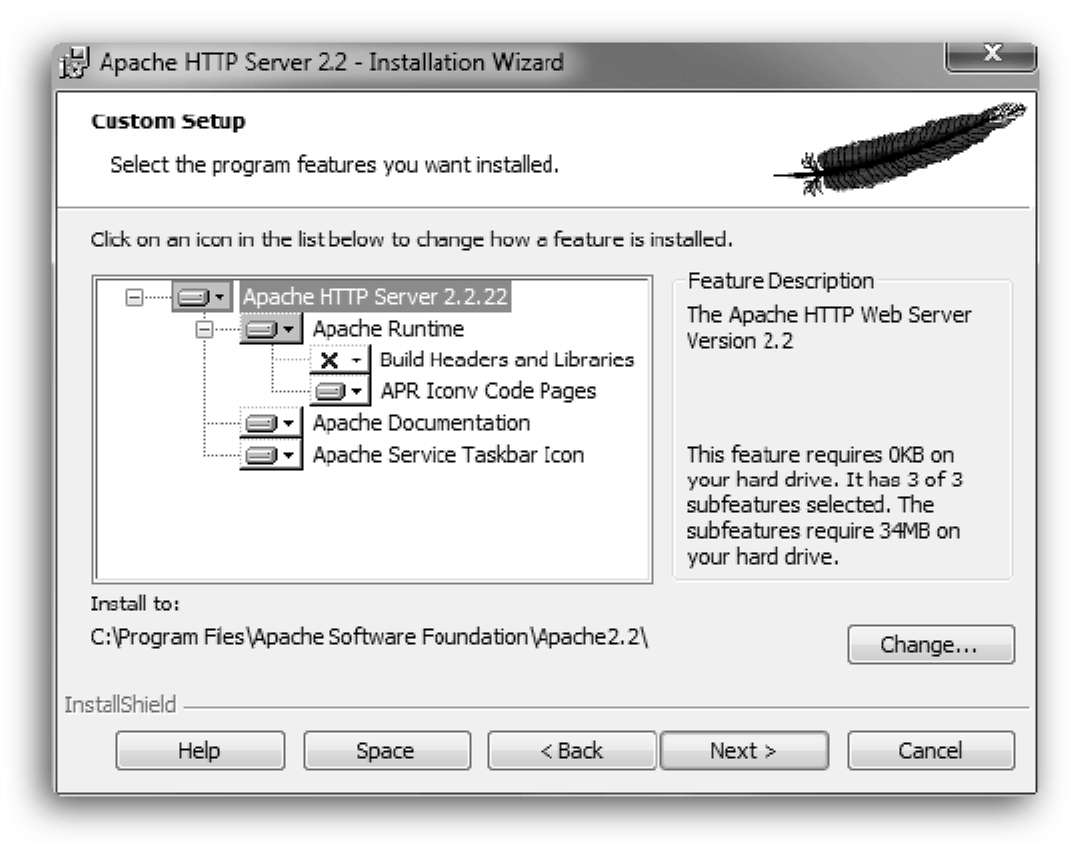
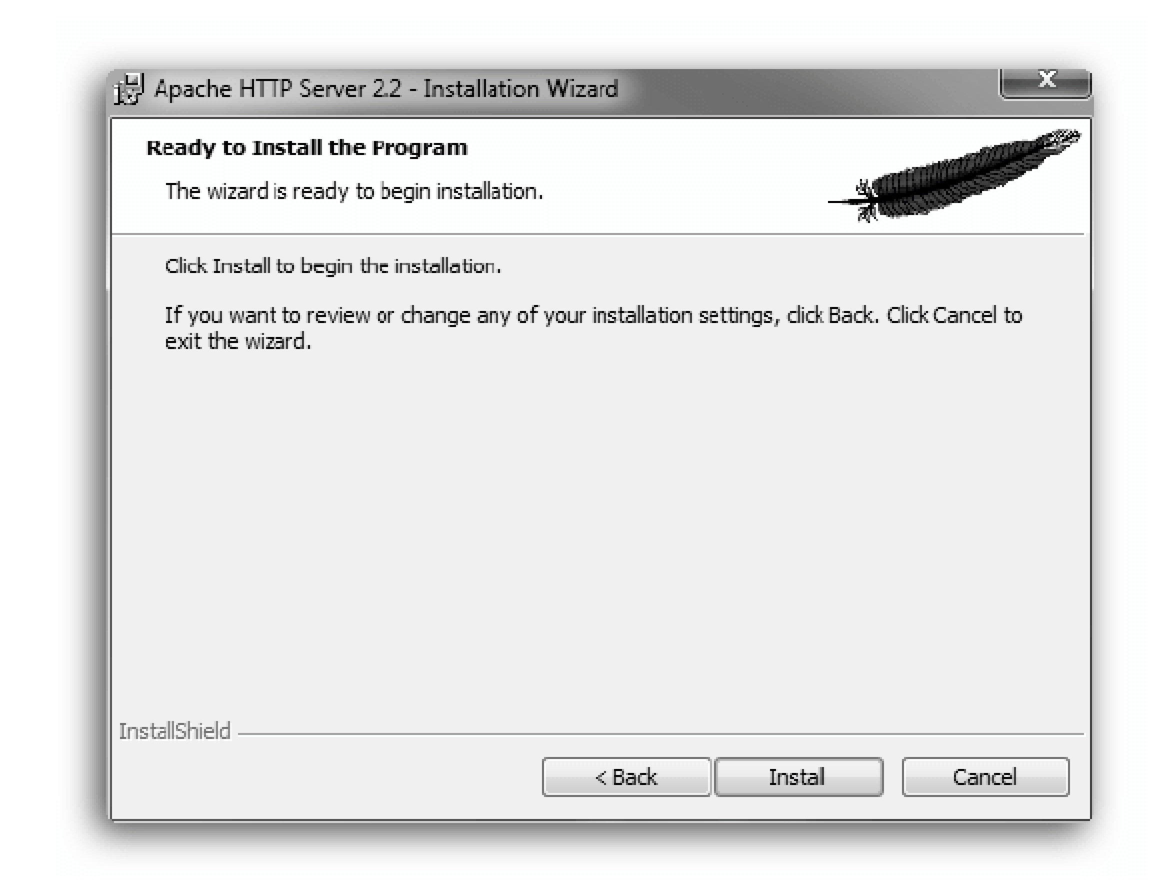


图 2-16　选择安装路径

在如图 2-16 所示的“自定义安装（Custom Setup）”界面，单击“修改（Change…）”按钮修改安装路径。然后单击“下一步（Next）”按钮，弹出“开始安装（Ready to Install the Program）”对话框，如图 2-17 所示。此时，单击“安装（Install）”按钮，安装程序便会按照上面的设置将 Apache HTTP 服务器安装到指定的目录里。



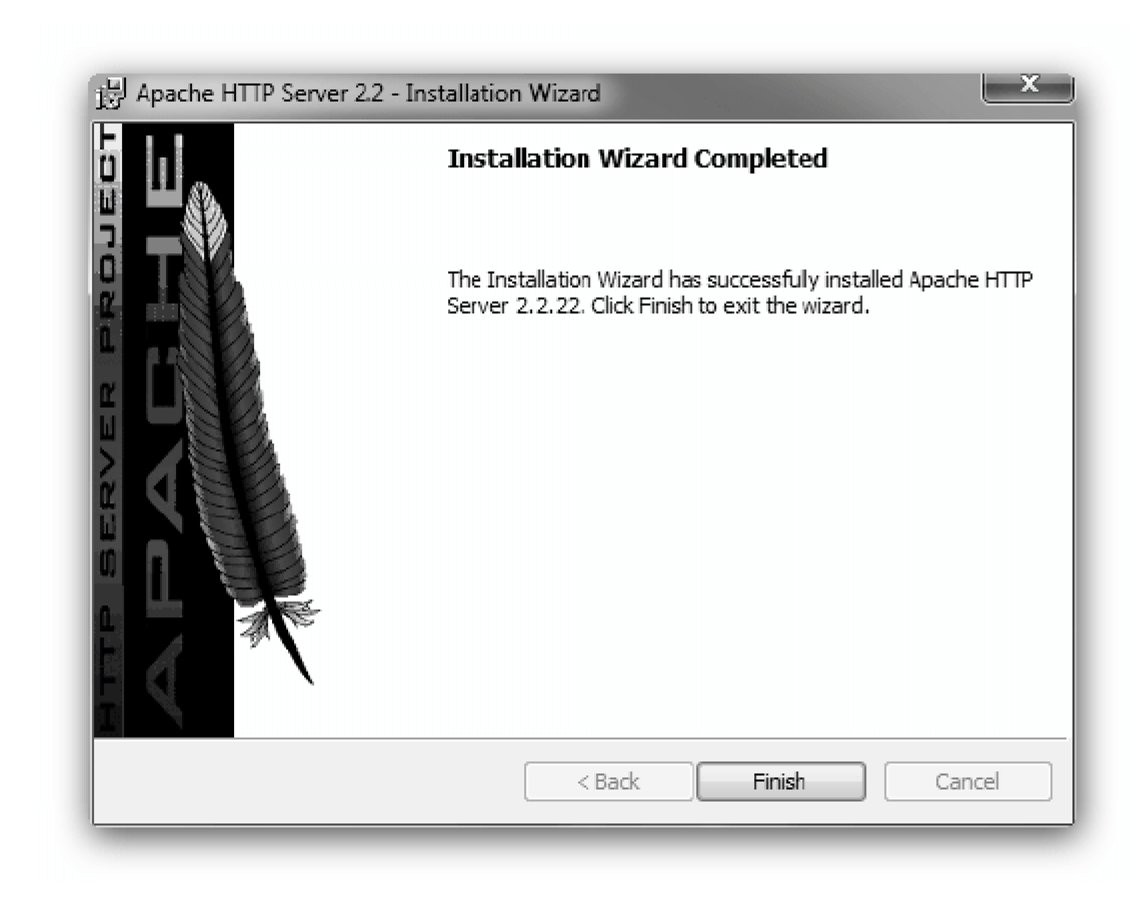




图 2-17　开始安装　　　　图 2-18　安装结束

当弹出如图 2-18 所示的对话框时，整个安装过程宣告结束。在执行安装时，可能会出现一些命令行窗口，无视即可。单击“结束（Finish）”按钮，关闭对话框。

2.2.2　安装验证

在安装结束后，读者可以通过如下方法来验证 Apache HTTP 服务器安装是否成功：

- 检查系统托盘中是否出现了图标。若出现了该图标，表示 Apache HTTP 服务器安装成功并正在运行。若系统托盘中出现的是图标，Apache HTTP 服务器安装成功，但并未运行，此时单击开始菜单，找到 Apache HTTP 服务器的文件夹。然后打开“控制 Apache HTTP 服务器（Control Apache HTTP Server）”文件夹，单击“启动（Start）”按钮。若启动成功，请进行下一步的验证。若启动失败，则需要调整 Apache 服务器提供服务的端口号。
- 打开浏览器，在地址框内输入 http://localhost 并按回车键。若出现如图 2-19 所示的页面，表示 Apache 服务器运行正常。

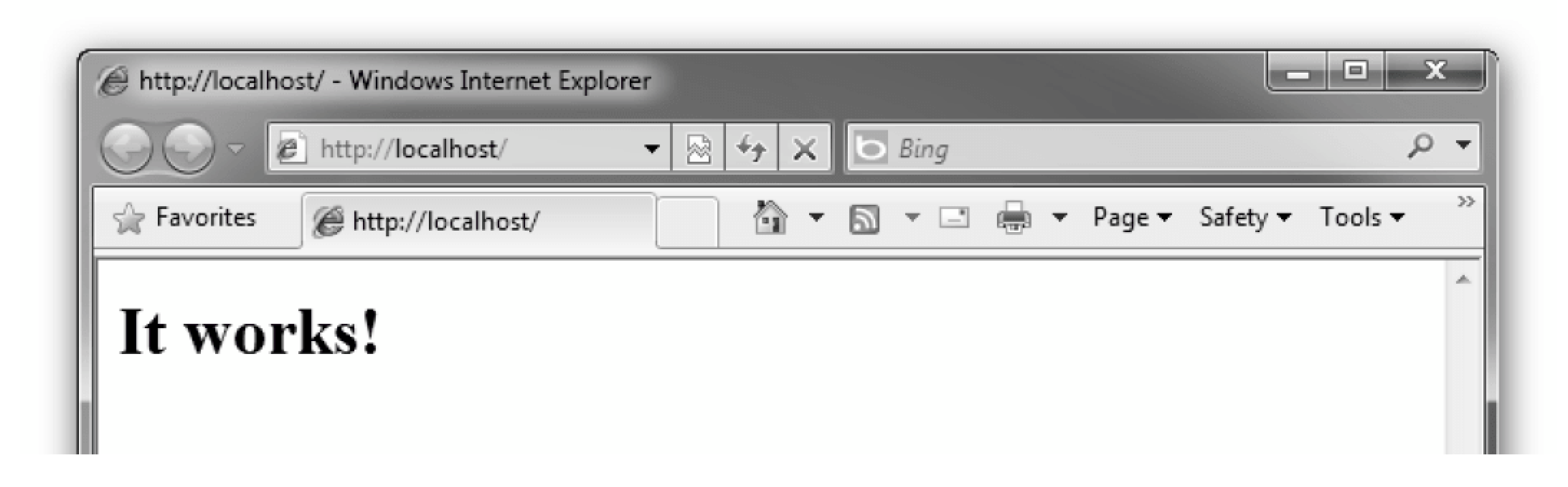


图 2-19　Apache 服务器安装成功并正常运行

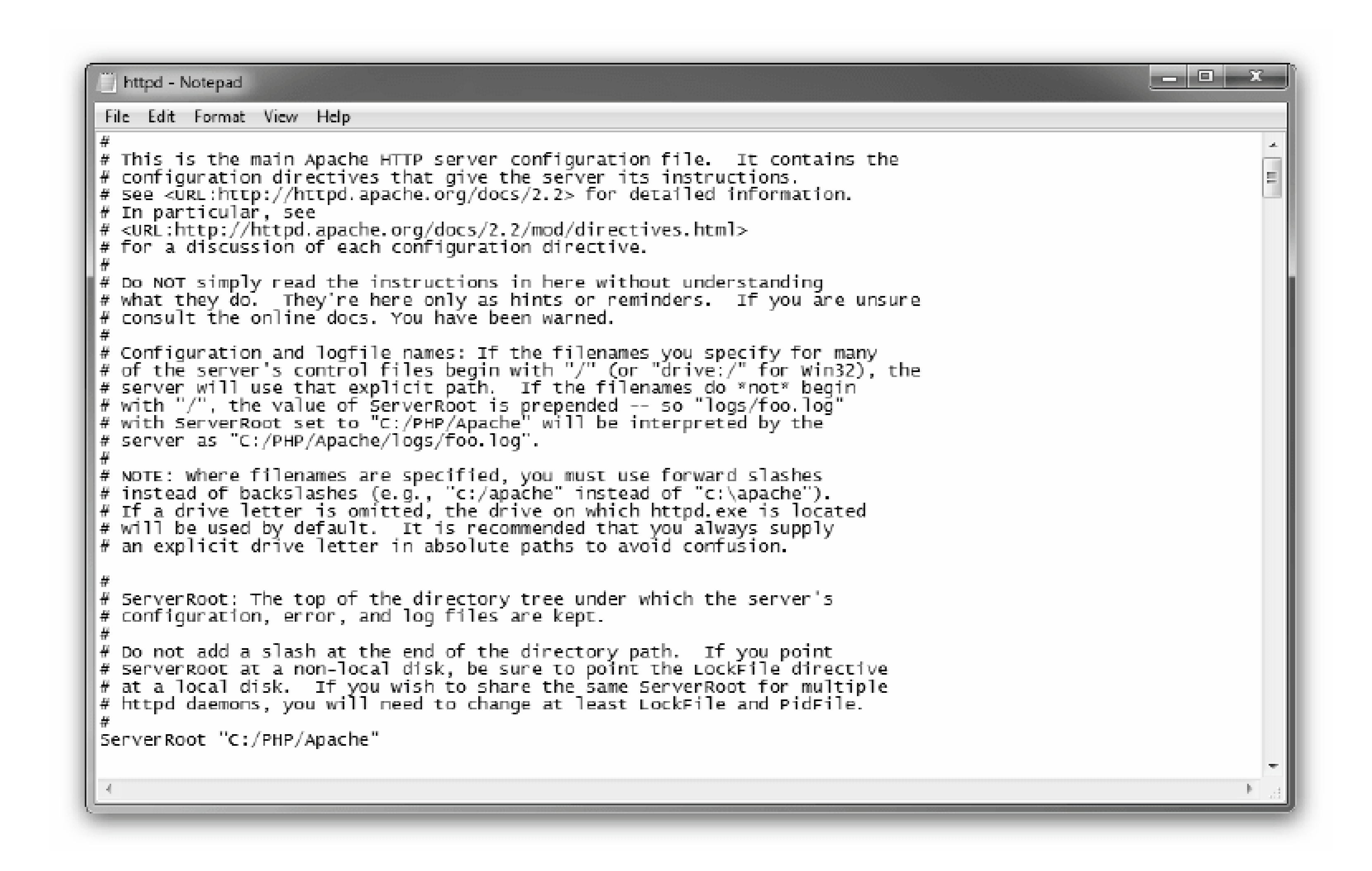
2.2.3　配置 Apache HTTP 服务器

完成 Apache HTTP 服务器的安装之后，需要对其进行简单的配置，以使其更好地和 PHP 引擎互动。

单击屏幕左下角的开始按钮，单击“所有程序（All Programs）”|“Apache HTTP 服务器 2.2（Apache HTTP Server 2.2）”|“配置 Apache 服务器（Configure Apache Server）”|“编辑 Apache httpd.conf 配置文件（Edit the Apache httpd.conf Configuration File）”选项，打开 httpd.conf 文件，如图 2-20 所示。

在文件中，需要找如下几行代码：

```
ServerRoot "C:/PHP/Apache"                  //服务器安装目录
…
Listen 80                                   //服务器监听端口
…
DocumentRoot "C:/PHP/Apache/htdocs"         //服务器文件目录
…
<Directory "C:/PHP/Apache/htdocs">          //服务器文件目录
```


```
#
# This is the main Apache HTTP server configuration file.  It contains the
# configuration directives that give the server its instructions.
# See <URL:http://httpd.apache.org/docs/2.2> for detailed information.
# In particular, see
# <URL:http://httpd.apache.org/docs/2.2/mod/directives.html>
# for a discussion of each configuration directive.
#
# Do NOT simply read the instructions in here without understanding
# what they do.  They're here only as hints or reminders.  If you are unsure
# consult the online docs. You have been warned.
#
# Configuration and logfile names: If the filenames you specify for many
# of the server's control files begin with "/" (or "drive:/" for Win32), the
# server will use that explicit path.  If the filenames do *not* begin
# with "/", the value of ServerRoot is prepended -- so "logs/foo.log"
# with ServerRoot set to "C:/PHP/Apache" will be interpreted by the
# server as "C:/PHP/Apache/logs/foo.log".
#
# NOTE: Where filenames are specified, you must use forward slashes
# instead of backslashes (e.g., "c:/apache" instead of "c:\apache").
# If a drive letter is omitted, the drive on which httpd.exe is located
# will be used by default.  It is recommended that you always supply
# an explicit drive letter in absolute paths to avoid confusion.

#
# ServerRoot: The top of the directory tree under which the server's
# configuration, error, and log files are kept.
#
# Do not add a slash at the end of the directory path.  If you point
# ServerRoot at a non-local disk, be sure to point the LockFile directive
# at a local disk.  If you wish to share the same ServerRoot for multiple
# httpd daemons, you will need to change at least LockFile and PidFile.
#
ServerRoot "C:/PHP/Apache"
```

图 2-20　Apache httpd.conf 文件截图

找到这些代码后，请确认：

- 服务器的安装目录是否为之前安装 Apache HTTP 服务器时选择的目录。在这里，Apache HTTP 服务器路径为 C:/PHP/Apache。
- 服务器的监听端口是否为 80。若在“2.2.2　安装验证”小节中发现 Apache HTTP 服务器无法正常启动，可尝试将服务器的监听端口改成 8080 或其他端口，再重启 Apache HTTP 服务器。

在确认过 Apache 服务器的安装目录和监听端口后，需要修改服务器的文件目录。在本书中，服务器的文件目录的路径为 C:/PHP/htdocs/。修改后的 httpd.conf 文件如下：

```
ServerRoot "C:/PHP/Apache"                //服务器安装目录
…
Listen 80                                 //服务器监听端口
…
DocumentRoot "C:/PHP/ htdocs"             //修改后的服务器文件目录
…
<Directory "C:/PHP/ htdocs">              //修改后的服务器文件目录
```

修改完成后，请重启 Apache HTTP 服务器。

2.3　安装和配置 PHP 脚本处理引擎

由于获取的 PHP 引擎包是 ZIP 格式的压缩文件，因此我们需要先将其解压缩，然后再完成相关配置。

2.3.1　解压 PHP 引擎包

如图 2-21 所示，右键单击之前获取的 PHP 引擎包文件，在弹出的快捷菜单中选择“解压（Extract All…）”，弹出如图 2-22 所示的对话框。

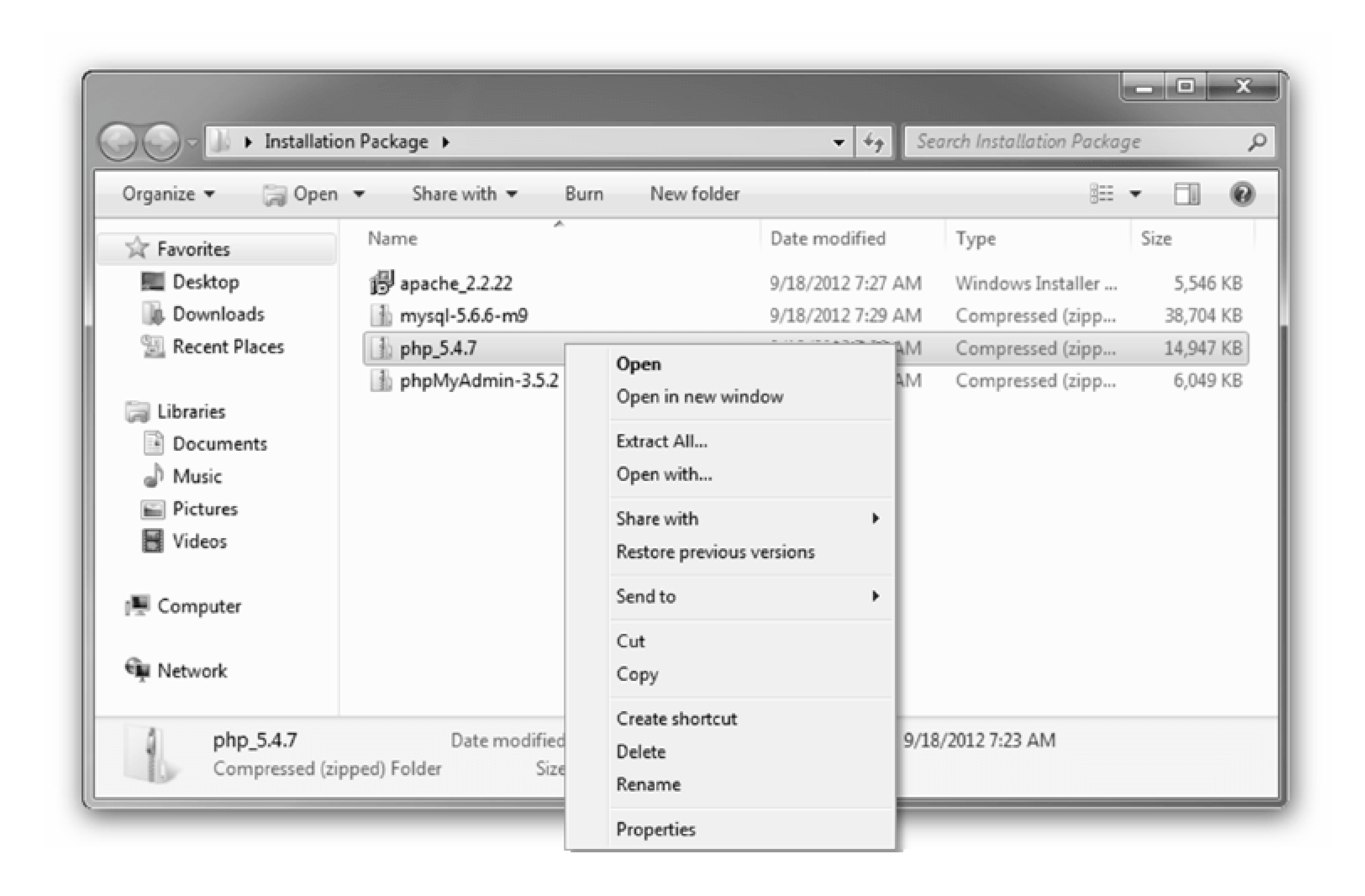


图 2-21　解压 PHP 引擎包

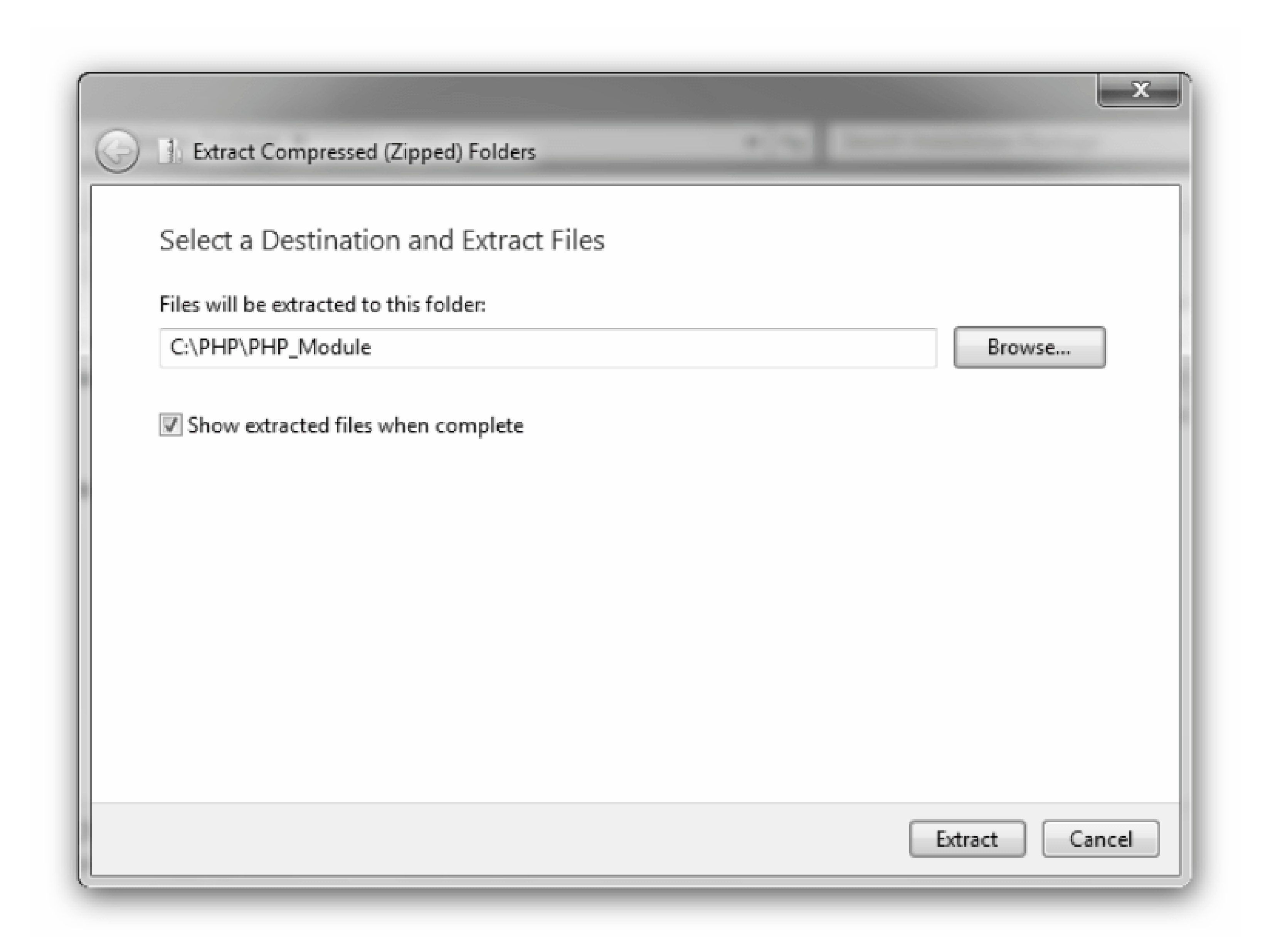


图 2-22　选择解压目标路径

单击“浏览（Browse）”按钮，将路径修改为 C:/PHP/PHP_Module，然后单击“解压（Extract）”按钮。解压完成后 PHP_Module 目录自动打开并显示在桌面上。

2.3.2　配置 PHP 引擎

在使用 PHP 引擎之前，我们需要介绍 Apache 服务器和 PHP 引擎互相认识。因此，这部分的配置，我们分两步走。第一步，介绍 PHP 引擎给 Apache 服务器认识，也就是配置 Apache 服务器的 httpd.conf 文件；第二步，介绍 Apache 服务器给 PHP 引擎认识，也就是配置 PHP 引擎的 php.ini 文件。通过这两个步骤，Apache 和 PHP 引擎就互相认识了对方并成了一对好搭档。

1．配置Apache服务器的httpd.conf文件

打开 Apache 服务器的 httpd.conf 文件。找到“Dynamic Shared Object (DSO) Support”，然后在一连串的 LoadModule 后加上一行：

```
LoadModule php5_module "C:/PHP/PHP_Module/php5apache2_2.dll"
```

然后找到如下代码：

```
<IfModule dir_module>
   DirectoryIndex index.html
</IfModule>
```

将其修改为：

```
<IfModule dir_module>
   DirectoryIndex index.html index.php     //将 index.php 设置为默认首页文件
</IfModule>
```

然后找到如下代码：

```
AddType application/x-gzip .tgz .gz
```

在其后添加：

```
AddType application/x-httpd-php.php
AddType application/x-httpd-php.html
```

随后保存修改好的 httpd.conf 文件。

2．配置PHP引擎的php.ini文件

当 PHP 引擎包解压完成后，我们会在 C:/PHP/PHP_Module 目录下发现两个 php.ini 文件，一个为 php.ini_dist，另一个为 php.ini_recommanded。这里我们将后者重命名为 php.ini，然后用记事本打开该文件，如图 2-23 所示。

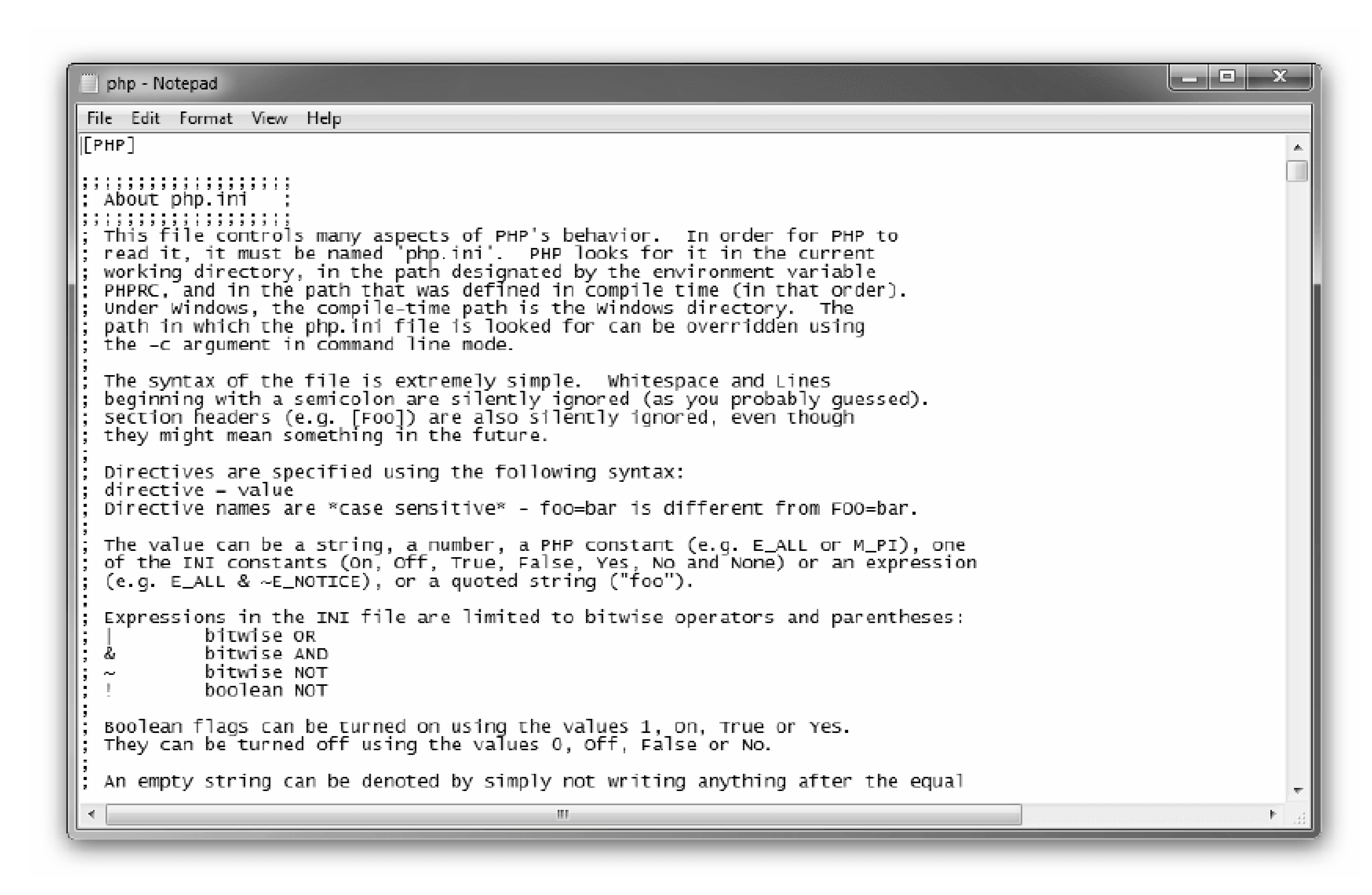


图 2-23　PHP 引擎的 php.ini 文件

在文件中搜索如下几行代码：

```
doc_root = ""                           //服务器文件根目录
…
extension dir = "./"                    //扩展库文件目录
…
;session.save_path = "/tmp"             //会话暂存目录
```

然后把它们修改为：

```
doc_root = " C:\PHP\htdocs"//将服务器文件根目录设置为 C:\PHP\htdocs
…
extension_dir = "C:\PHP\PHP_Module\ext"
                           //将扩展库文件目录设置为 C:\PHP\PHP_Module\ext
…
session.save path = " c:\php\session temp"
                           //将会话暂存目录设置为 c:\php\session_temp
```

接下来，在文件中搜索如下几行代码：

```
;extension=php_gd2.dll          //GD 库的扩展文件，用于在线生成、修改图片
…
;extension=php_gettext.dll      //PHP 的 gettext()函数扩展库
…
;extension=php_mbstring.dll     //多字节字符串处理模块扩展库
…
;extension=php_mysql.dll        //PHP 用于连接 MySQL 数据库的扩展库
…
;extension=php mysqli.dll       //PHP 用于连接 MySQL 数据库的扩展库
…
;extension=php_xmlrpc.dll       //RPC 扩展库，启用后，可以使用 XML 传输命令和数据
```

代码行前的分号（;）表示这些代码行被注释掉了，也就是说服务器在加载 PHP 引擎时，不会加载这些扩展库。如果需要加载这些扩展库，就需要将这些代码行前的分号（;）去掉。上面列出的扩展库只是在进行 PHP 脚本开发时，经常会用到的一些扩展库。读者也可以根据需要，自行开启其他的扩展库。

修改完成后，保存该 php.ini 文件，然后将其复制到 C:\Windows 路径下。之后，再将 PHP_Module 目录下的 libmysql.dll 和 libmcrypt.dll 文件复制到 C:\Windows\System32 目录下。随后重启 Apache HTTP 服务器。在重启完成后， Apache 服务器就算是和 PHP 引擎相互认识了。

2.3.3　配置验证

在配置完成后，需要验证一下配置是否正确。为此，需要先编写一个测试文件，并将其命名为 index.php，然后通过浏览器访问这个文件。

测试文件的内容如下：

```
<?php
phpinfo();
?>
```

打开记事本，将上述代码复制到记事本中。然后单击“文件（File）”|“另存为（Save

As…）”按钮，弹出如图 2-24 所示的对话框。



图 2-24　将文件保存为 index.php

在“文件类型（Save As type）”下拉框中单击“所有文件（All Files）”按钮，在“文件名（File name）”文本框中填写 index.php，然后单击“保存”按钮，将文件保存在服务器文件根目录（C:\PHP\htdocs）里。

打开 Internet Explorer 浏览器，在地址栏中输入 http://localhost，然后回车查看结果。如果浏览器显示如图 2-25 所示的页面，说明 PHP 配置成功。

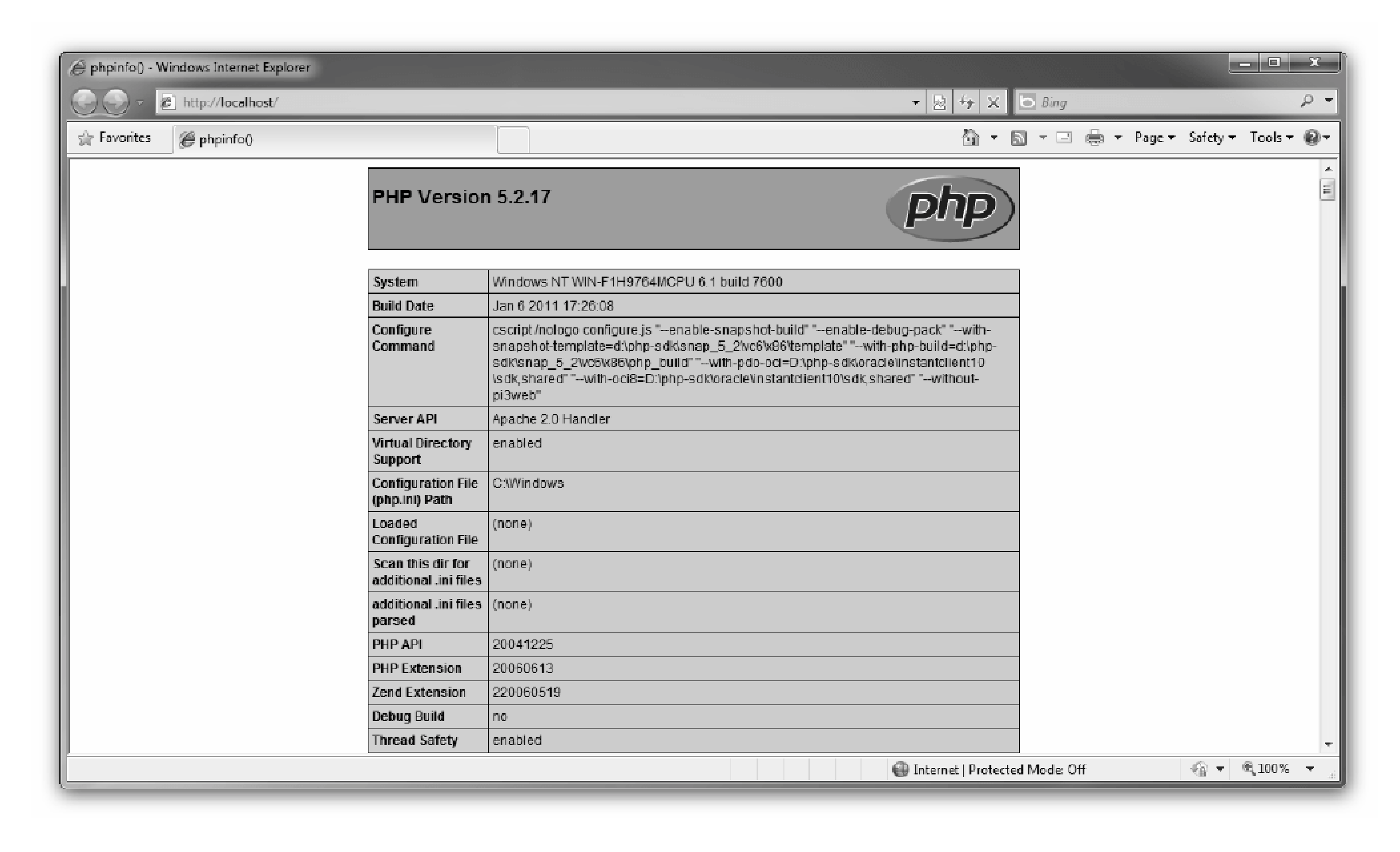


图 2-25　PHP 版本信息

2.4　安装和配置 MySQL 数据库

在完成 Apache 服务器和 PHP 引擎的安装与配置之后，需要继续安装和配置 MySQL 数据库，以便为正在搭建的 PHP 平台提供强大的数据库支持。需要注意的是，MySQL 数据库的安装需要有 Microsoft .NET 4.0 支持。若读者的操作系统中没有安装.NET 4.0，请到微软的官方网站下载。

2.4.1　安装 MySQL 数据库

双击在 2.1.3 小节中获取的 MySQL 数据库安装包，弹出如图 2-26 所示的对话框。



图 2-26　MySQL 数据库安装界面

单击“安装 MySQL 产品（Install MySQL Products）”链接后的效果如图 2-27 所示。选中“我接受这份许可证协议中的条款（I accept the license terms）”选项，然后单击“下一步（Next）”按钮。



图 2-27　MySQL 许可证协议

因为下载的是最新稳定版本的 MySQL 软件，因此在随后出现的“获取软件最新更新（Find latest products）”页面中，如图 2-28 所示，选中“跳过更新检查（Skip the check for updates）”选项，然后单击“执行（Execute）”按钮。

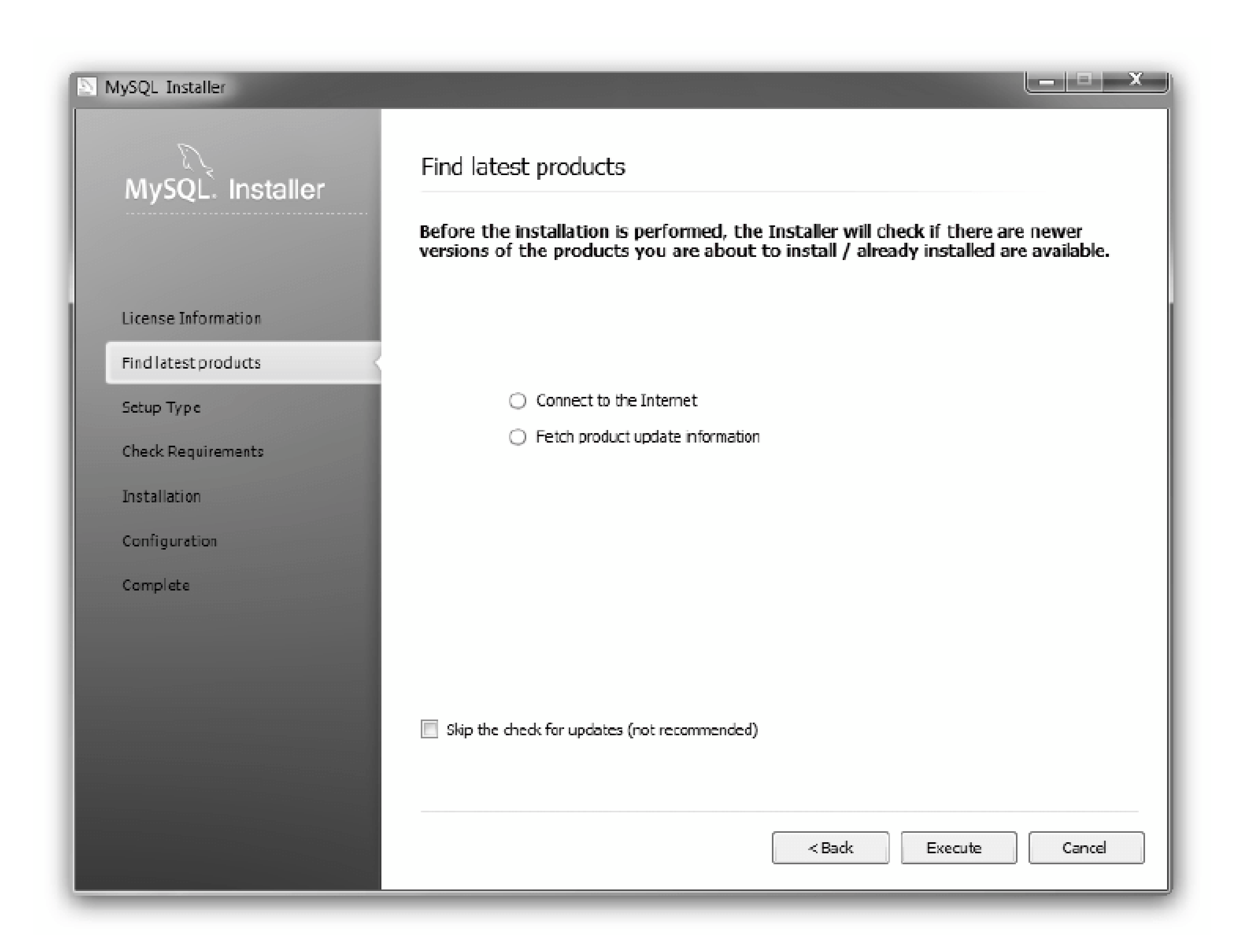


图 2-28　获取软件最新更新

更新检查分为两步，第一步将会检查 Internet 连接，若检查通过，则会获取产品更新信息。读者也可以直接单击“执行（Execute）”按钮来获取最新更新。

在接下来的页面里，需要选择安装类型。如图 2-29 所示，一共有五种安装类型。它们分别是开发者（默认）、仅服务器、仅客户端、完全安装和自定义。

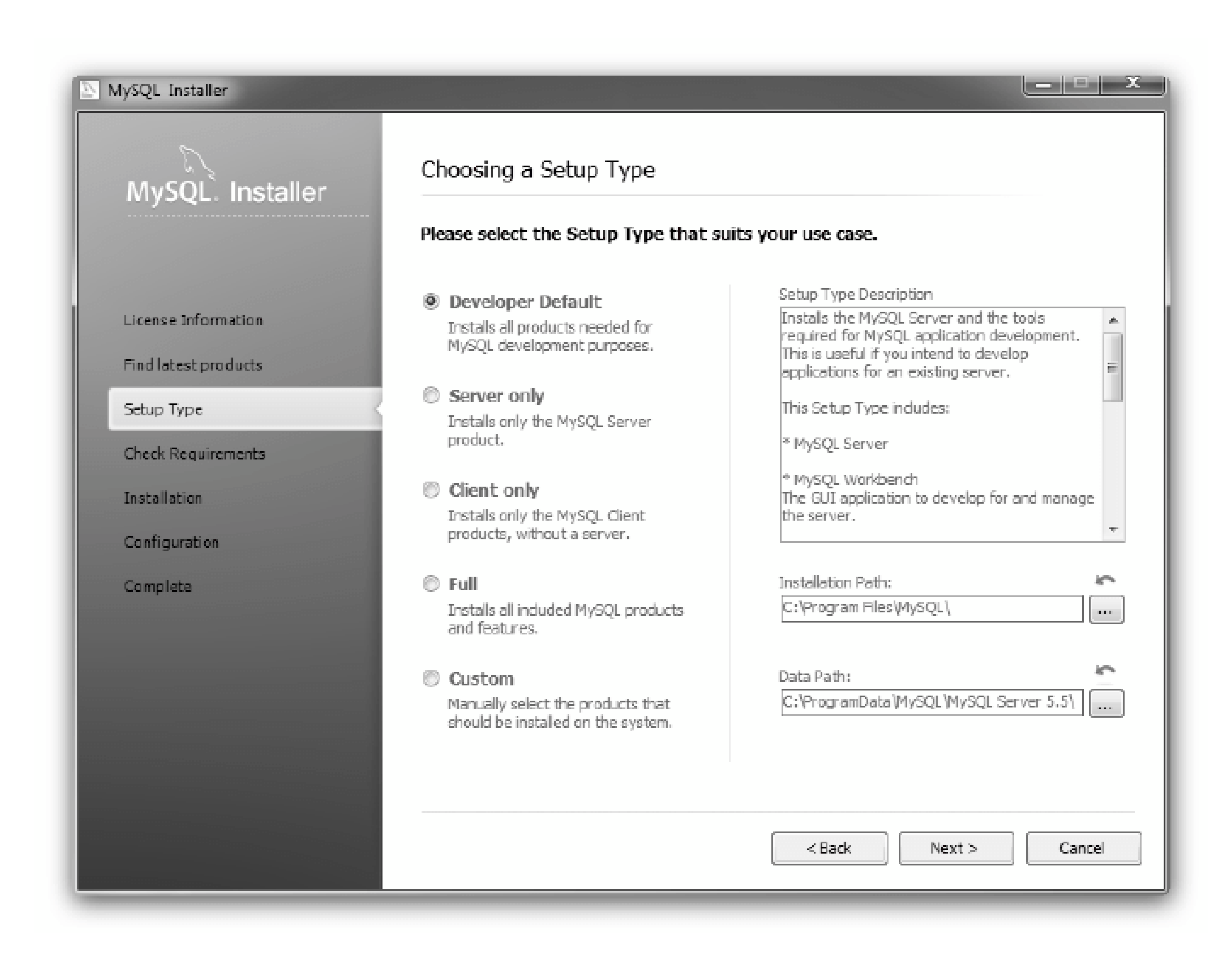


图 2-29　选择安装类型

- 选择开发者（默认），将会安装进行 MySQL 相关开发所需的文件。
- 选择仅服务器，将且仅会安装 MySQL 服务器程序。

- ❑ 选择仅客户端，将且仅会安装 MySQL 客户端程序。
- ❑ 选择完全安装，将会安装 MySQL 服务器和客户端程序。
- ❑ 选择自定义安装，读者可以根据自己的需要选择安装的内容。

按照之前的规划，phpMyAdmin 会被用来管理 MySQL 数据库，因此，这里选择的安装方式为仅服务器。然后将“安装路径（Installation Path）”和“数据路径（Data Path）”分别修改为 C:\PHP\MySQL 和 C:\PHP\MySQL\MySQL Server 5.5\，然后单击“下一步（Next）”按钮。

在执行安装之前，系统会列出需要安装的项目，如图 2-30 所示。

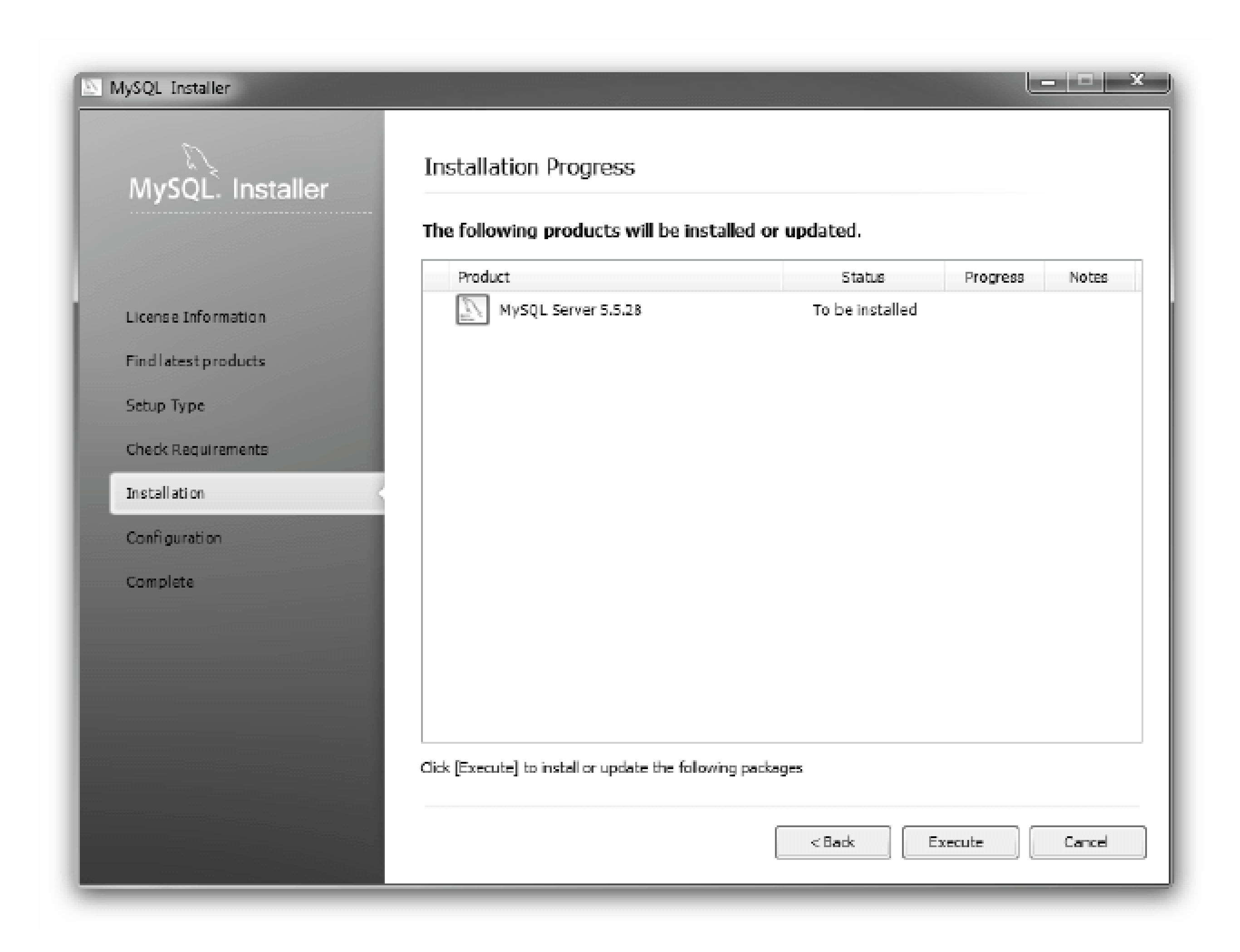


图 2-30　需要安装的项目

安装项目确认无误后，单击“执行（Execute）”按钮开始安装。安装完成后，确认列表中各安装项目的“状态（Status）”栏均显示为“安装成功（Install success）”，然后单击两次“下一步（Next）”按钮，进行 MySQL 服务器的配置界面，如图 2-31 所示。



图 2-31　配置 MySQL 服务器

保持默认配置即可。直接单击“下一步（Next）”按钮。在随后弹出如图 2-32 所示的界面上，可以设置 root 账户的密码和添加新的数据库用户。在这里，建议只设置 root 账户密码，不添加新的数据库用户。



图 2-32　设置 root 账户的密码和添加新的数据库用户

设置完成后，单击“下一步（Next）”按钮，弹出如图 2-33 所示的界面。在此，读者可以设置 MySQL 的服务名称。推荐保持默认设置即可。



图 2-33　设置 MySQL 服务名

单击“下一步（Next）”按钮，结束 MySQL 服务器的配置。

2.4.2　配置验证

当完成了上面的配置之后，还得需要验证一下配置是否正确。

首先，单击开始菜单，选择“所有程序（All Programs）”|“MySQL”|“MySQL Server 5.5”命令，弹出如图 2-34 所示的 MySQL 5.5 命令行客户端（MySQL 5.5 Command Line Client）对话框。



图 2-34　MySQL 5.5 命令行客户端

按照系统提示，输入之前设置的密码，然后回车，进入 MySQL 5.5 系统提示符界面，如图 2-35 所示。

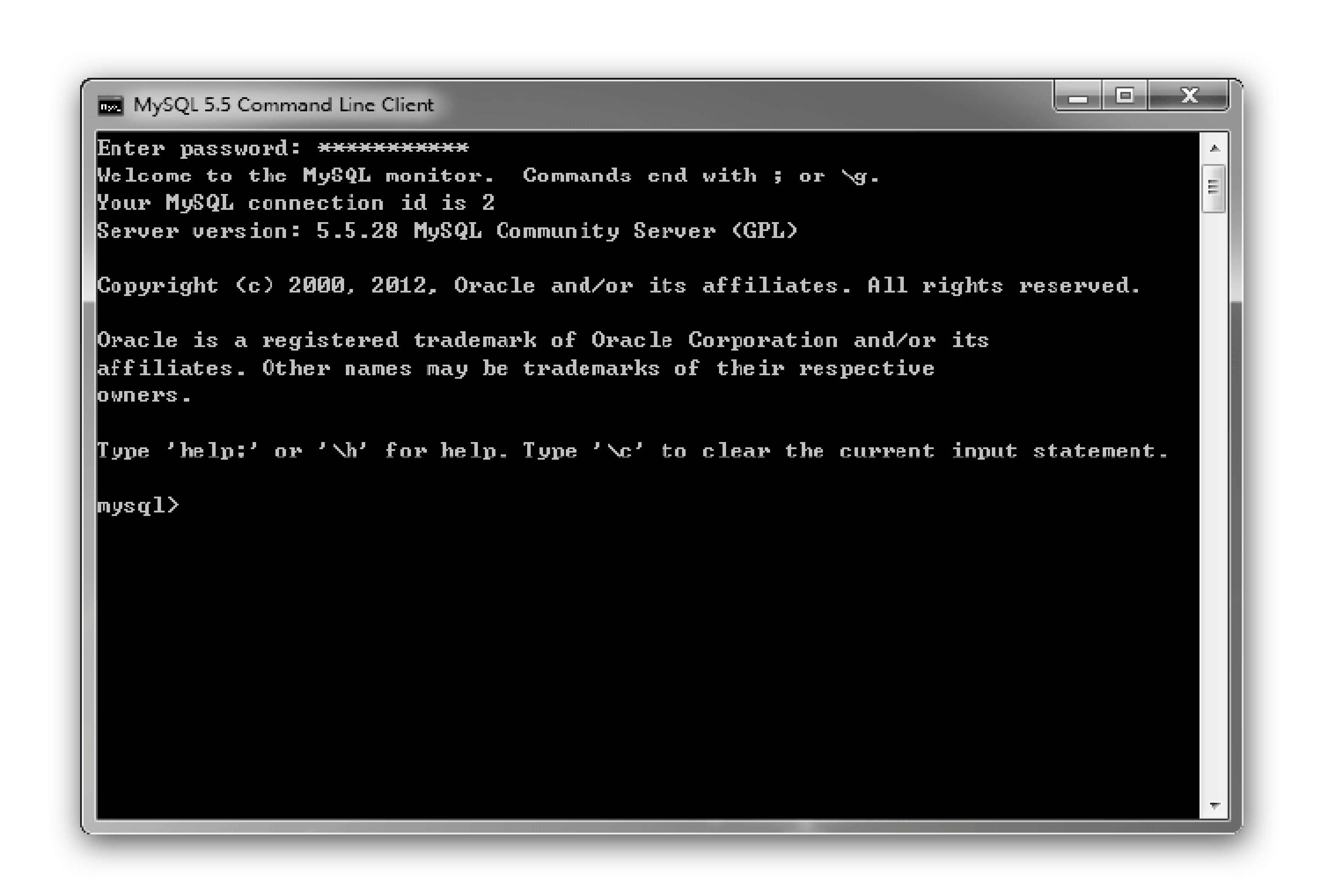


图 2-35　MySQL 5.5 系统命令提示符

在系统提示符界面，可以看到诸如 MySQL 连接号、MySQL 服务器版本、版权信息及命令行帮助信息。若需要通过命令行操作 MySQL 数据库软件，请参阅 MySQL 服务器的产品文档。

2.5　安装和配置 phpMyAdmin

其实当安装和配置过程进行到这里，就可以收工了，因为 Apache HTTP 服务器+PHP 脚本处理引擎+MySQL 数据库的环境就算是搭建起来了。换句话说，如果读者知道如何通过命令行来管理数据库，那么这一节的内容就可以忽略掉了。

但是，读者也同样错过了一个认识到 PHP 强大能力的机会。在获取 phpMyAdmin 的时候，我们曾经介绍过，phpMyAdmin 是一款完全由 PHP 脚本写的数据库管理软件。因此，phpMyAdmin 的源代码是认识并学习 PHP 和数据库交互的最好教材。那么闲话少说，现在就来了解一下如何安装和配置这款基于 Web 的 MySQL 数据库管理软件吧。

2.5.1　解压 phpMyAdmin 压缩包

在之前获取的 phpMyAdmin 压缩包上右键单击，选择“解压所有（Extract All...）”命令。弹出如图 2-36 所示的对话框。单击“浏览（Browse）”按钮修改解压路径为 C:\PHP\htdocs\phpMyAdmin3，然后单击“解压（Extract）”按钮。

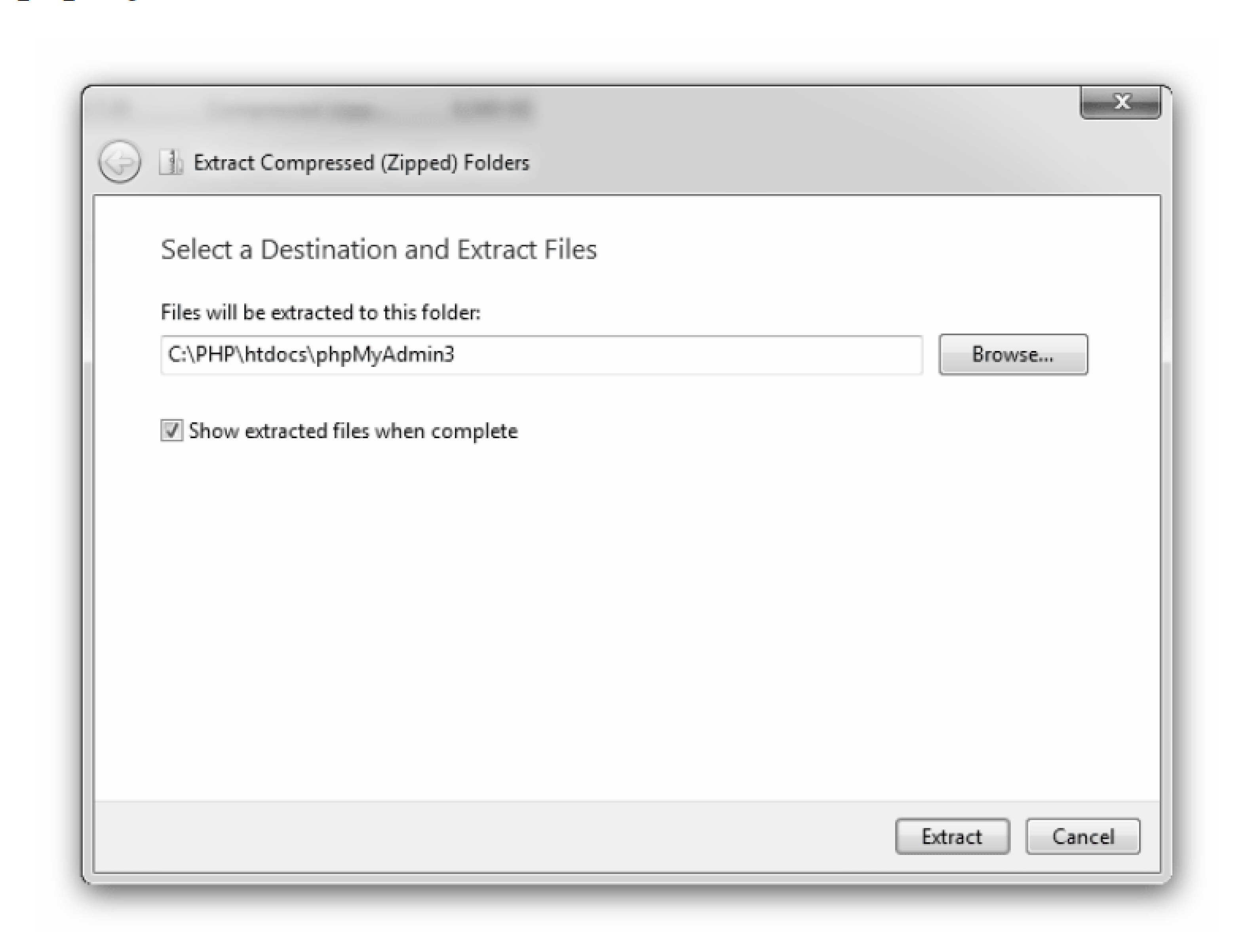


图 2-36　解压 phpMyAdmin 压缩包到指定目录

解压完成后，phpMyAdmin 目录自动打开并显示在桌面上。这样，就算是完成了 phpMyAdmin 压缩包的解压工作。

2.5.2　配置 phpMyAdmin

在浏览器地址栏中输入 http://localhost/phpmyadmin3/setup，然后按键盘上的回车键，会弹出如图 2-37 所示的配置界面。



图 2-37　phpMyAdmin 安装配置界面

（1）根据“服务器”区域框内的提示信息显示，系统中没有配置好的服务器。单击“新建服务器”按钮，在弹出如图 2-38 所示的页面中新建一个服务器。

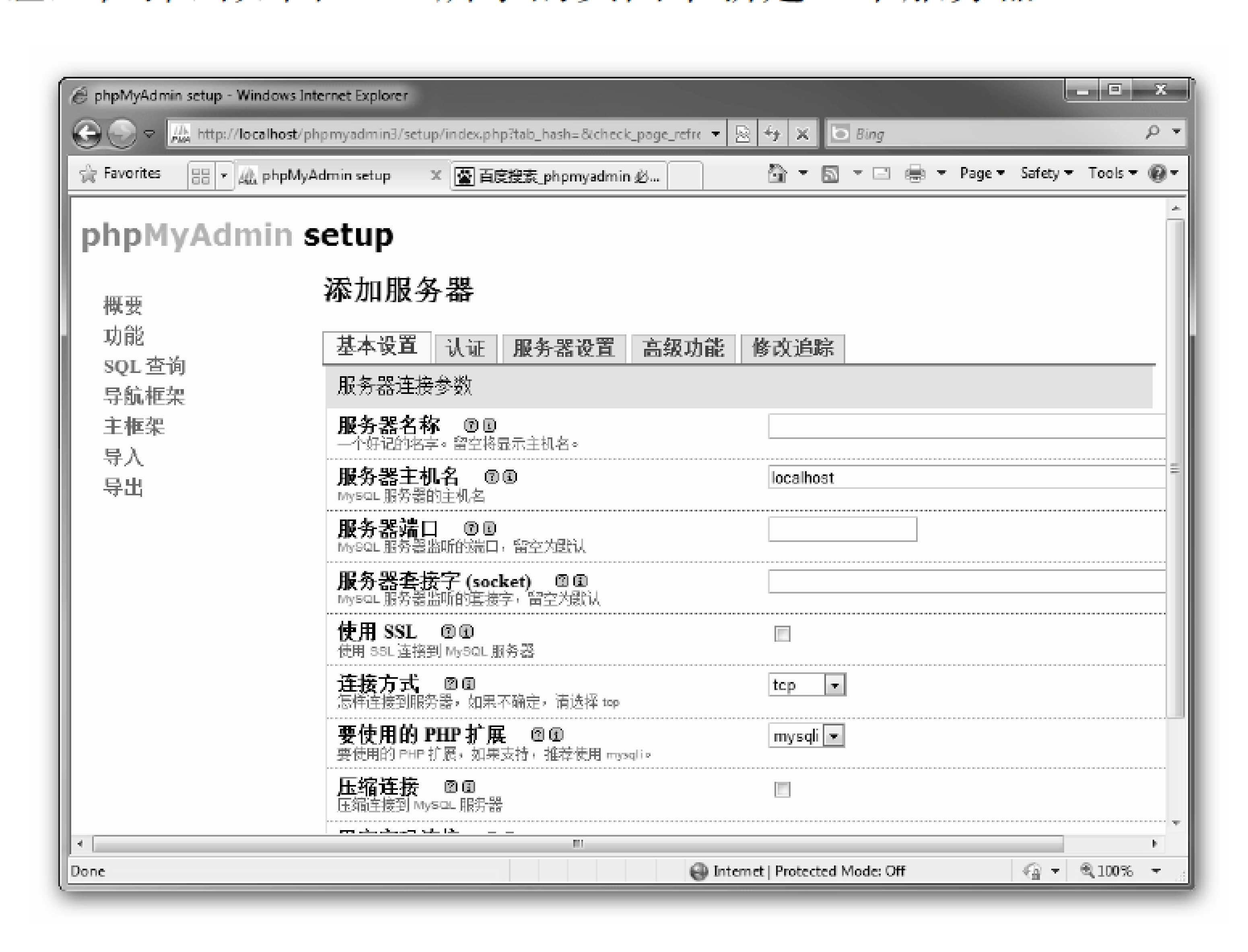


图 2-38　添加服务器

（2）直接单击该页面下方的“保存”按钮，系统会返回到安装首页，并弹出如图 2-39 所示的提示信息。

按照系统的提示，可以使用 SSL 来加密与数据库服务器的连接。不过在本书中，数据库服务器就是本地计算机，可以不使用 SSL 连接。另外，系统自动为我们设置了一个短语密码用于 Cookies 加密。到此，服务器就添加完成了。

（3）在提示信息下方的“服务器”区域框中，可以看到已经添加的服务器相关信息。在“配置文件”区域框中，可以看到需要保存的配置信息。确认无误后，单击“保存”按钮，如图 2-40 所示。



图 2-39　完成添加服务器后出现的系统提示信息



图 2-40　已添加的服务器相关信息和需要保存的配置文件信息

（4）当系统弹出如图 2-41 所示的提示时，配置文件保存成功。在按照提示信息将配置文件从 config 文件夹中复制到 phpMyAdmin 的根目录之后，就可以使用 phpMyAdmin 来管理 MySQL 数据库了。

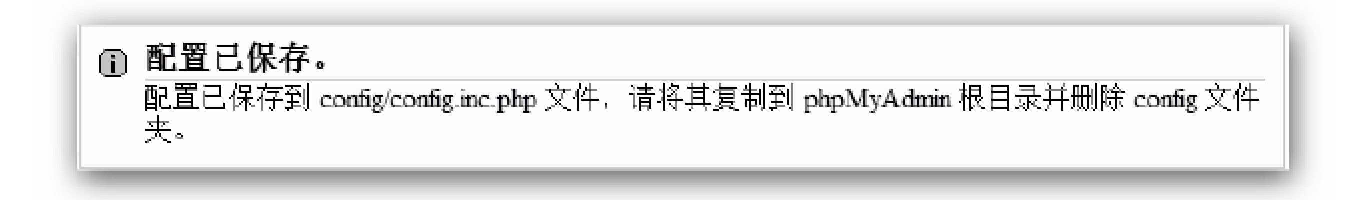


图 2-41　配置文件保存成功

在浏览器中输入 http://localhost/phpmyadmin3，然后按键盘上的回车键，弹出如图 2-42 所示的登录界面。这时，可以使用 root 用户登录 phpMyAdmin 系统。有同学可能会问，密码是什么啊？密码就是你在配置 MySQL 服务器时设置的那个 MySQL Root Password。不记得的同学面壁吧。

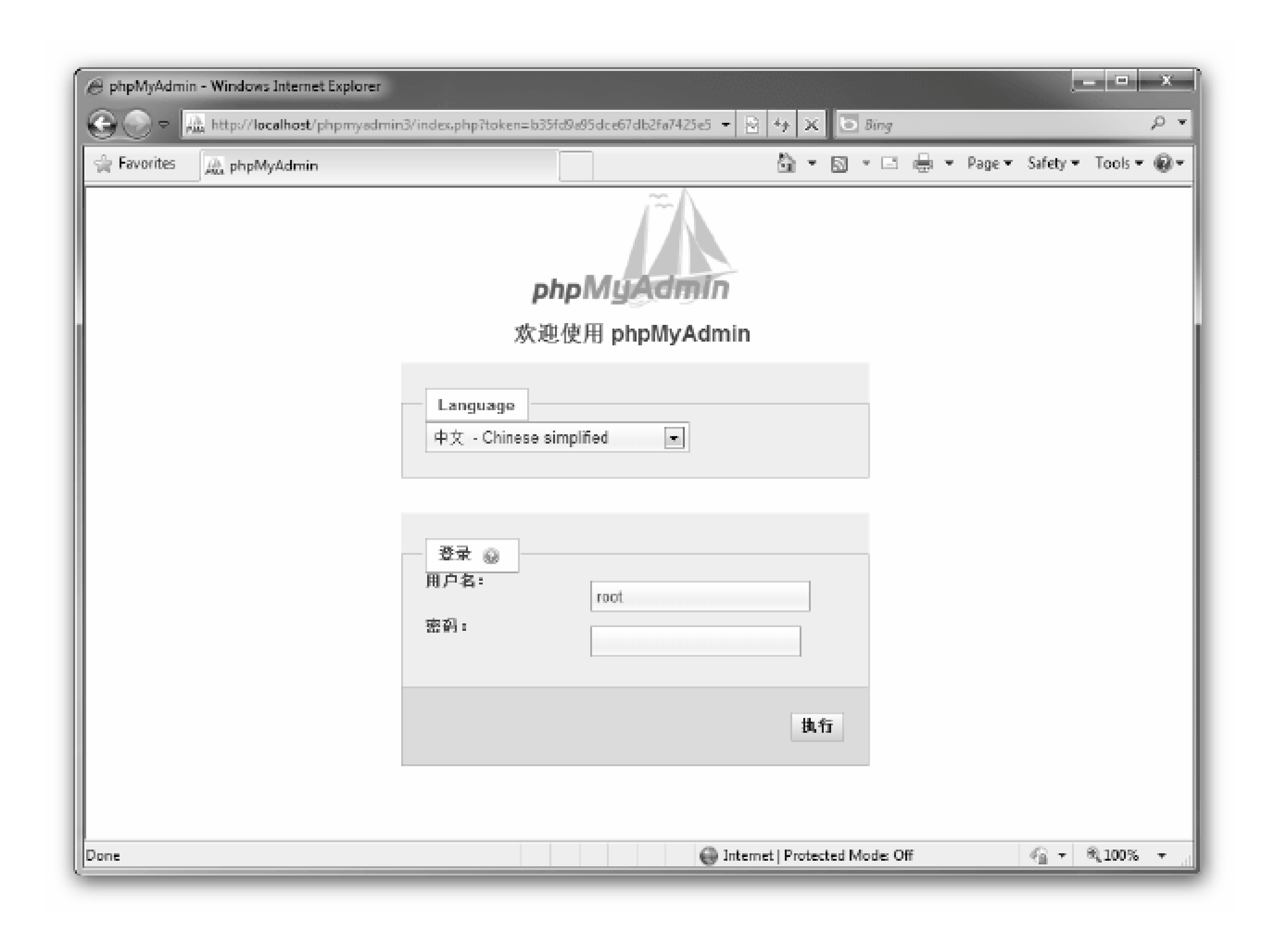


图 2-42　phpMyAdmin 登录页面

2.6　使用套件包搭建 PHP 运行环境

有的同学看到这里，可能已经头昏眼花、汗流浃背了。有的同学甚至会感叹道：“搭建个环境都这么复杂，那正儿八经地学起 PHP 脚本语言来会不会更难啊？”

为了提高大家搭建 PHP 运行环境的效率，有很多的大牛开发了不同版本的一键安装包，而且这些安装包差不多都是免费的。本书只介绍两个：一个为国产软件 PHPnow，另一个为常用的 WampServer。大家可以按照个人的需要进行选择。需要注意的是，两者都只能安装在 32 位的 Windows 操作系统中。

2.6.1　PHPnow

按照 PHPnow 官方网站的介绍，PHPnow 是一款 Windows 32 位操作系统下绿色、免费的 Apache+PHP+MySQL 环境套件包。使用该套件包可以方便快捷地搭建支持虚拟主机的 PHP 环境。套件包附带 PnCp.cmd 控制面板，帮助用户快速地配置套件，使用非常方便。

在这一小节里，我们就看看如何下载、安装和配置 PHPnow 套件包。

（1）访问 PHPnow 官方网站（http://www.phpnow.org），下载 PHPnow 的最新套件包。解压后双击如图 2-43 所示的 setup.cmd 文件。

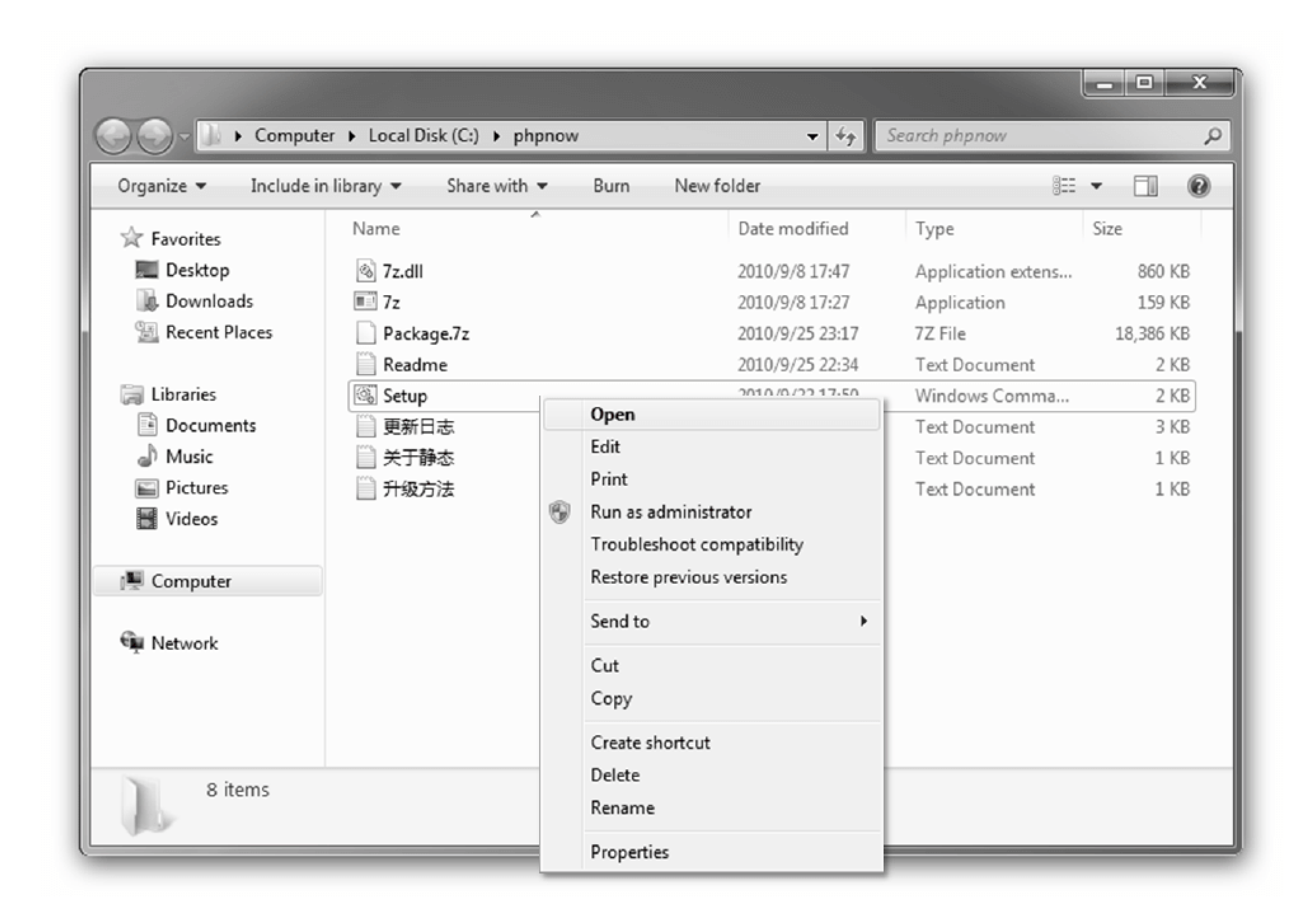


图 2-43　安装 PHPnow 套件包

（2）根据系统提示，我们需要选择 Apache 服务器版本和 MySQL 服务器版本系统，如图 2-44 所示。在输入所需版本后，系统会自动将所需文件解压到当前目录。

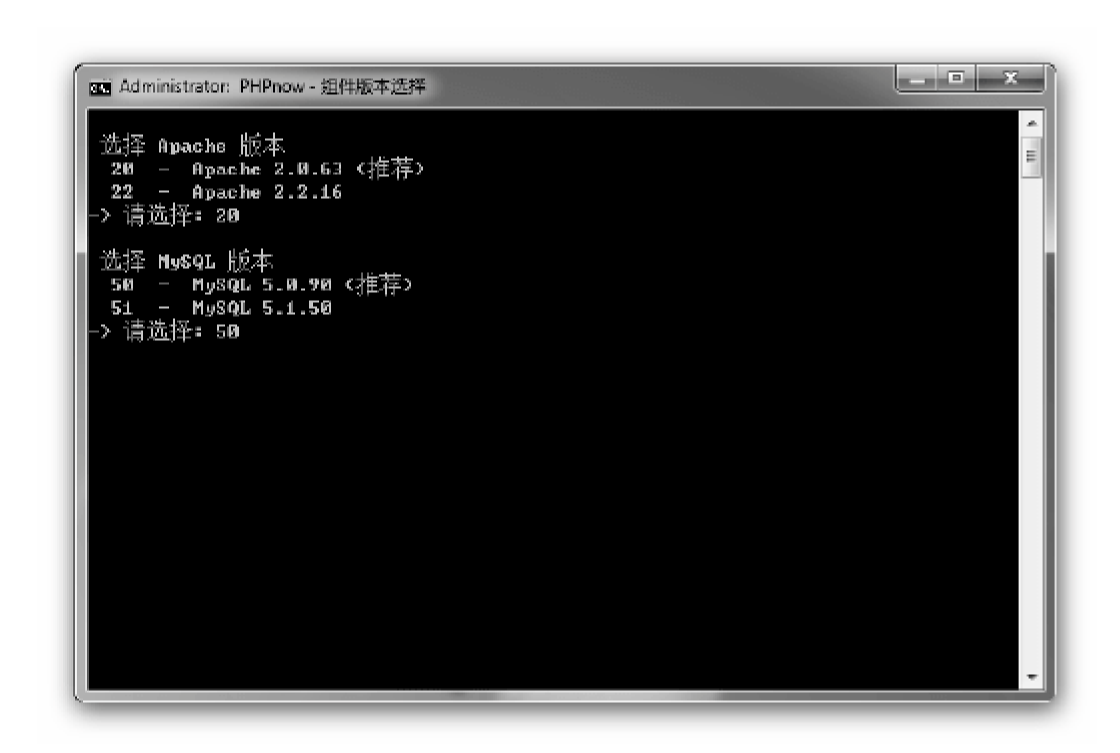


图 2-44　选择 Apache 和 MySQL 版本

（3）在解压完成后，系统会要求执行 Init.cmd 进行初始化，如图 2-45 所示。输入 Y 后，系统将启动 Apache 服务和 MySQL 服务，并要设置 MySQL 服务器 root 用户的密码，如图 2-46 所示。

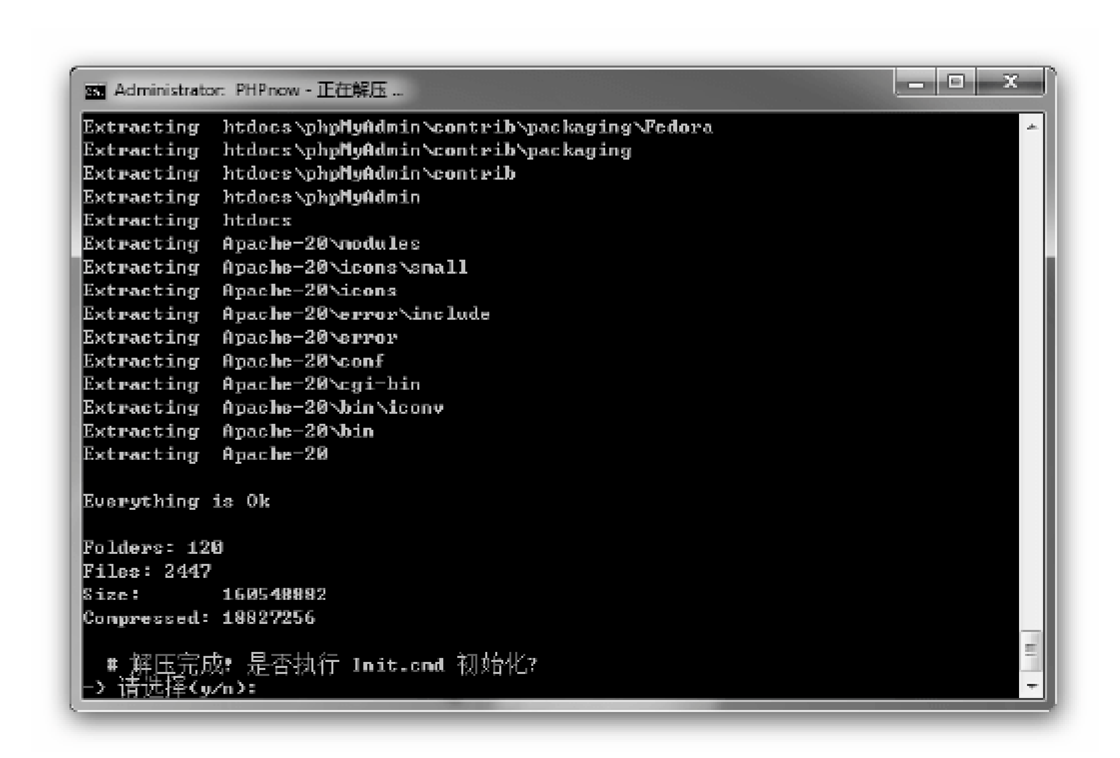




图 2-45　执行 Init.cmd 初始化　　　　　　　　　　图 2-46　设置 root 用户密码

（4）设置密码后，系统提示“全部完成！！你将可以看到 PHPnow 的默认页面！”如图 2-47 所示。按任意键关闭当前窗口。此时可以打开浏览器，并在地址栏中输入 http://localhost 来验证 Apache+PHP+MySQL 的环境是否搭建成功。如果在浏览器中看到如图 2-48 所示的页面时，说明环境搭建成功。





图 2-47 完成设置　　　　　　　　　　图 2-48　PHPnow 安装成功

（5）在此页面上，在“MySQL 连接测试”区域框内的“MySQL 用户密码”文本框内输入之前设置的 root 用户密码，就可以测试 MySQL 连接。若测试成功，页面显示结果如图 2-49 所示。

MySQL 测试结果	
服务器 localhost	OK (5.0.90-community-nt)
数据库 test	OK

图 2-49　MySQL 连接测试结果

（6）PHPnow 套件包还提供了 MySQL 数据库在线管理软件 phpMyAdmin。读者只需单击“服务器信息（Server Information）”区域框中的相应链接即可。

当确认 PHPnow 安装配置成功后，可以看到现在的 phpnow 文件夹的内容与之前相比发生了变化。一起来认识一下如图 2-50 所示的文件和文件夹。

在这个文件夹中：

- ❑ 文件 PnCp.cmd 是 PHPnow 的控制面板，如图 2-51 所示。双击该文件可以打开 PHPnow 控制面板，进而完成诸如新建、修改和删除虚拟主机之类的配置服务器的操作。
- ❑ 文件夹 htdocs 是 PHPnow 用来存放网页文件的服务器根目录。可以将所有的网页文件放在这个目录中，并通过 http://localhost 地址来访问它们。
- ❑ 文件夹 Apache-20、MySQL-5.0.90 和 php-5.2.14-Win32 包含了 Apache 服务器、MySQL 服务器和 PHP 引擎的相关文件。
- ❑ 文件夹 Pn 和 PnCmds 则包含了 PHPnow 的相关文件和命令。

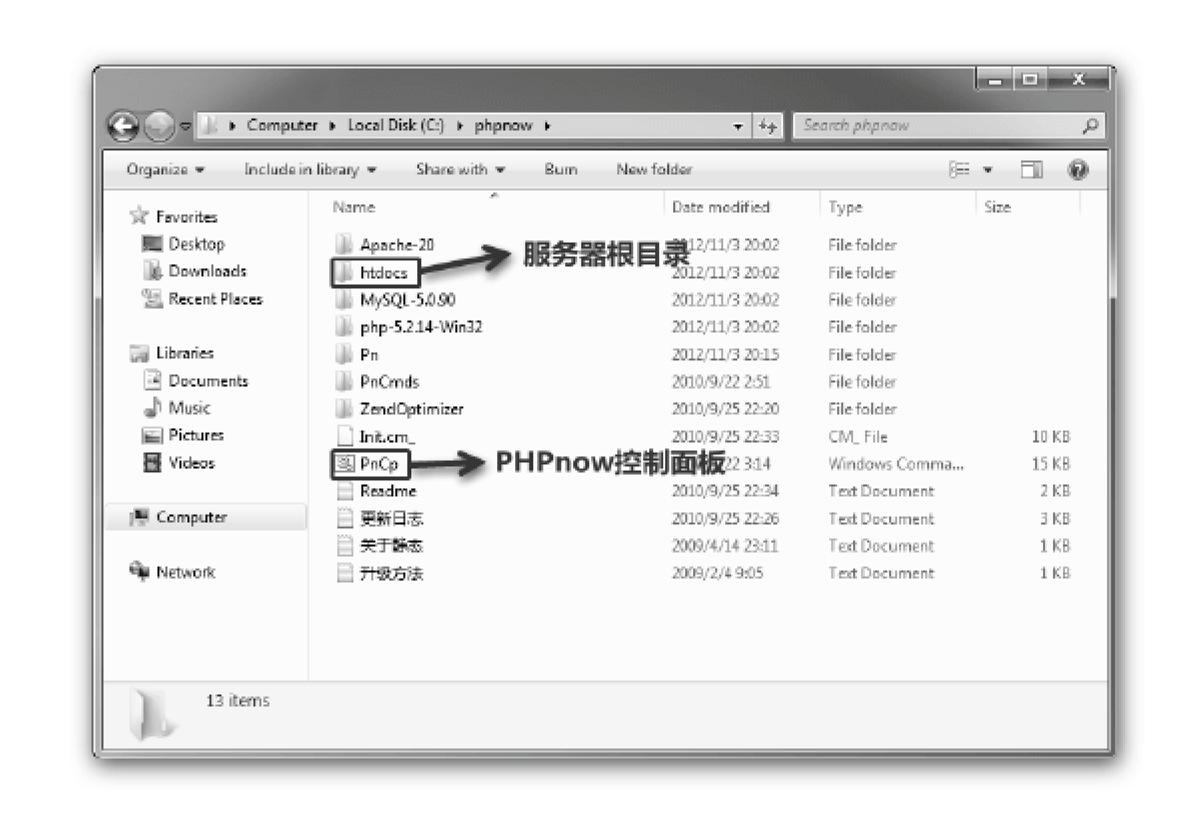


图 2-50　phpnow 文件夹



图 2-51　PHPnow 控制面板

2.6.2　WampServer

WampServer 包含的组件有：Apache 服务器、MySQL 服务器、PHP 处理引擎、phpMyAdmin 数据库管理软件和 Xdebug 组件。

在这一节里我们来看看如何下载、安装和配置 WampServer。

（1）访问 WampServer 官方网站，下载 WampServer 的安装包。下载完成后，双击安装包文本，开始安装和配置过程，如图 2-52 所示。单击“下一步（Next）按钮”。

（2）在如图 2-53 所示的页面中指定 WampServer 的安装路径，然后单击“下一步（Next）”按钮。安装完成过程中，系统会要求读者指定开发过程中需要使用的浏览器，直接单击“打开（Open）”按钮将使用 Internet Explorer 作为默认浏览器。



图 2-52　WampServer 安装界面

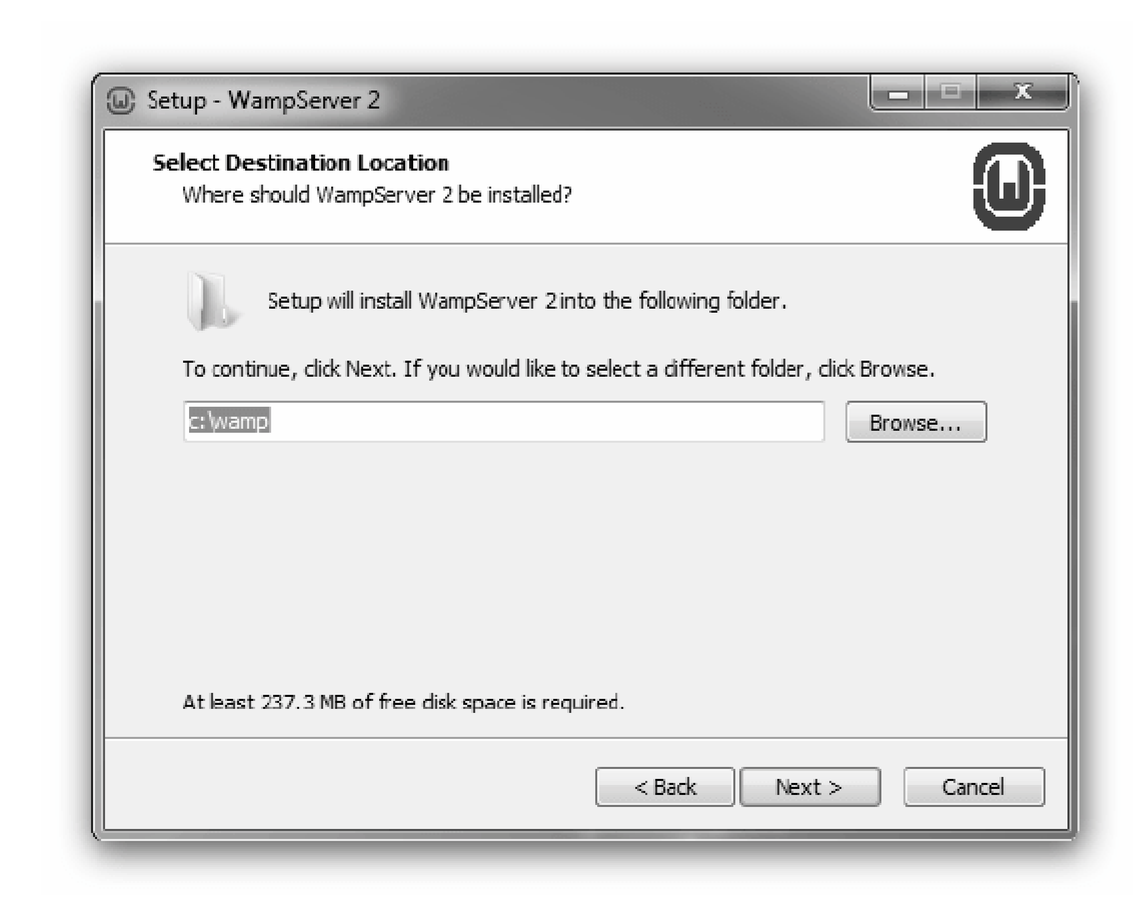


图 2-53　指定 WampServer 的安装路径

（3）安装完成后，检查系统托盘中的图标。若图标为绿色，表示服务器启动成功，若为橙色，服务器启动失败，请安装 VC++ 2010 运行库后再试。

（4）在确认服务器启动成功后，单击系统托盘中的，弹出如图 2-54 所示的快捷菜单。



图 2-54　WampServer 快捷菜单

通过此快捷菜单，可以根据自己的需要定制 Apache 服务器、PHP 引擎和 MySQL 服务器的相关设置。这里需要注意的是，MySQL 服务器的 root 用户密码为空。若有设置 root 用户密码的需要，请选择“MySQL”|“my.ini”命令，在弹出的文本文件中找到如下几行代码：

```
# The following options will be passed to all MySQL clients
[client]
#password   = your password
port          = 3306
socket      = /tmp/mysql.sock
```

然后将#password 一行前的#号去掉，然后在该行的等号后面输入需要的密码。确认无

误后，保存并关闭文件，最后重启 MySQL 服务即可。

2.7　在微软 IIS 上配置 PHP 运行环境

在第一章里，我们还提到 PHP 除了可以和 Apache 的 HTTP 服务器搭档之外，也可以与 Windows 操作系统自带的微软 IIS 服务器合作。本节将介绍如何让微软 IIS 服务器认识 PHP。

2.7.1　开启互联网信息服务

首先，我们需要知道的是，在 Windows 操作系统中自带有一个 Web 服务，名字叫做互联网信息服务（Internet Information Services）。但是，这一服务除了在服务器版的操作系统中是默认开启的以外，在其他版本的 Windows 操作系统中，都是默认关闭的。因此，第一步，需要启用互联网信息服务。

打开“控制面板”，找到并双击“程序和功能”图标，在弹出的窗口左侧找到“开启或关闭 Windows 功能”。然后，在弹出的“Windows 功能（Windows feature）”对话框中找到“互联网信息服务（Internet Information Services）”，单击其前方的复选框所示。然后展开“互联网信息服务”节点，找到并展开“万维网服务（World Wide Web Services）”节点，找到并展开“应用开发功能（Application Development Features）”节点，接着勾选该节点下的所有子项，如图 2-55 所示。

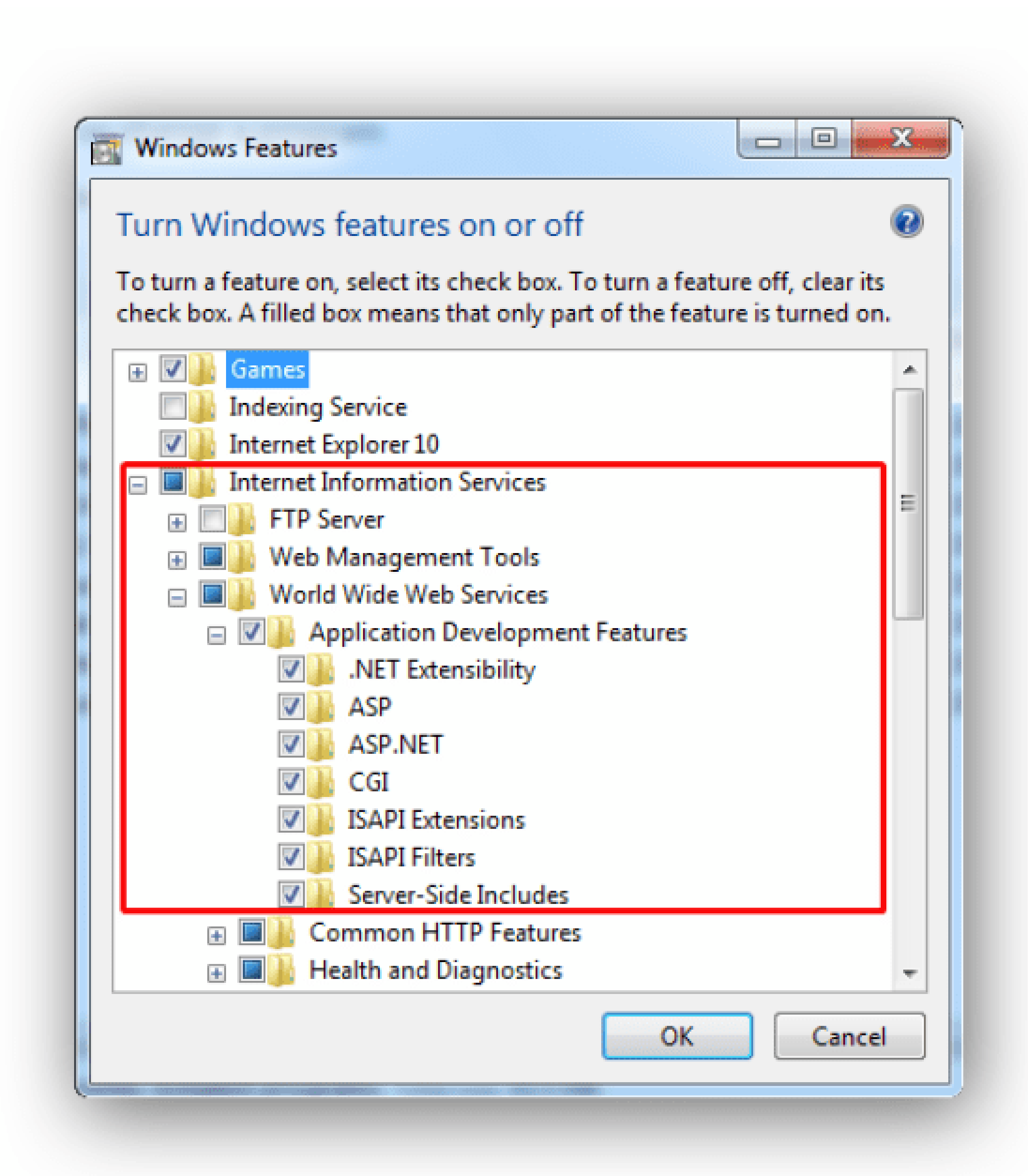


图 2-55　开启互联网信息服务（Internet Information Services）

在单击“确定”按钮关闭上述对话框后，回到控制面板。双击“管理工具（Administrative

Tools）”图标，看看里面是否存在“互联网信息服务管理器（Internet Information Services (IIS) Manager）”的快捷方式。若存在，表示 IIS 服务已经成功启用了。双击打开互联网信息服务管理器，会弹出如图 2-56 所示的窗口。

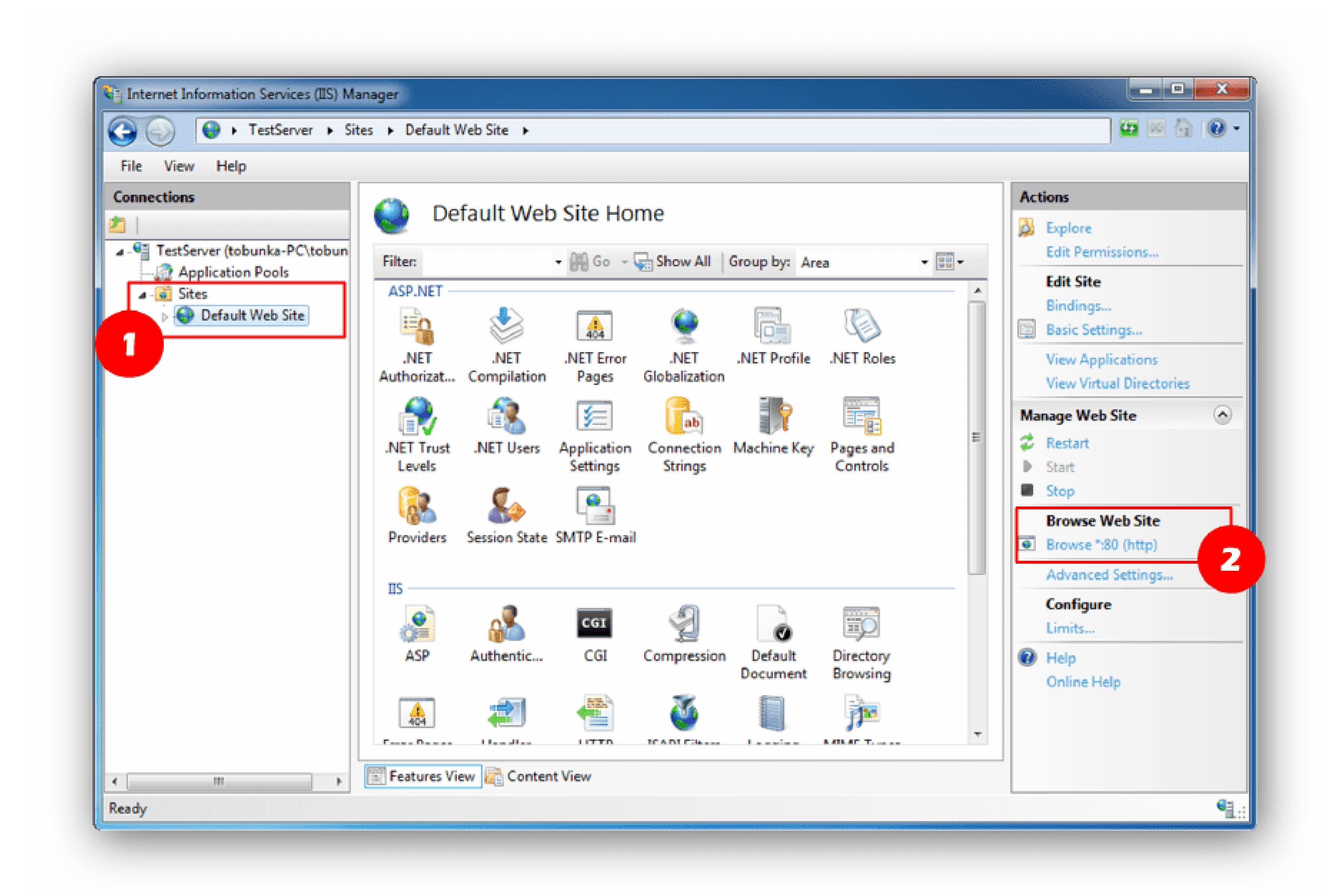


图 2-56　互联网信息服务管理器主界面

在此窗口中，可以看到很多的选项，不过大部分跟今天的任务无关。我们只需要展开窗口左侧导航树中的“站点（Sites）”，选中“默认站点（Default Web Site）”。然后单击窗口右侧的“浏览站点（Browser Web Site）”区域框内的“浏览*:80（http）”链接即可。在单击了该链接后，可以看到如图 2-57 所示的页面，表明 IIS 服务成功启用了。

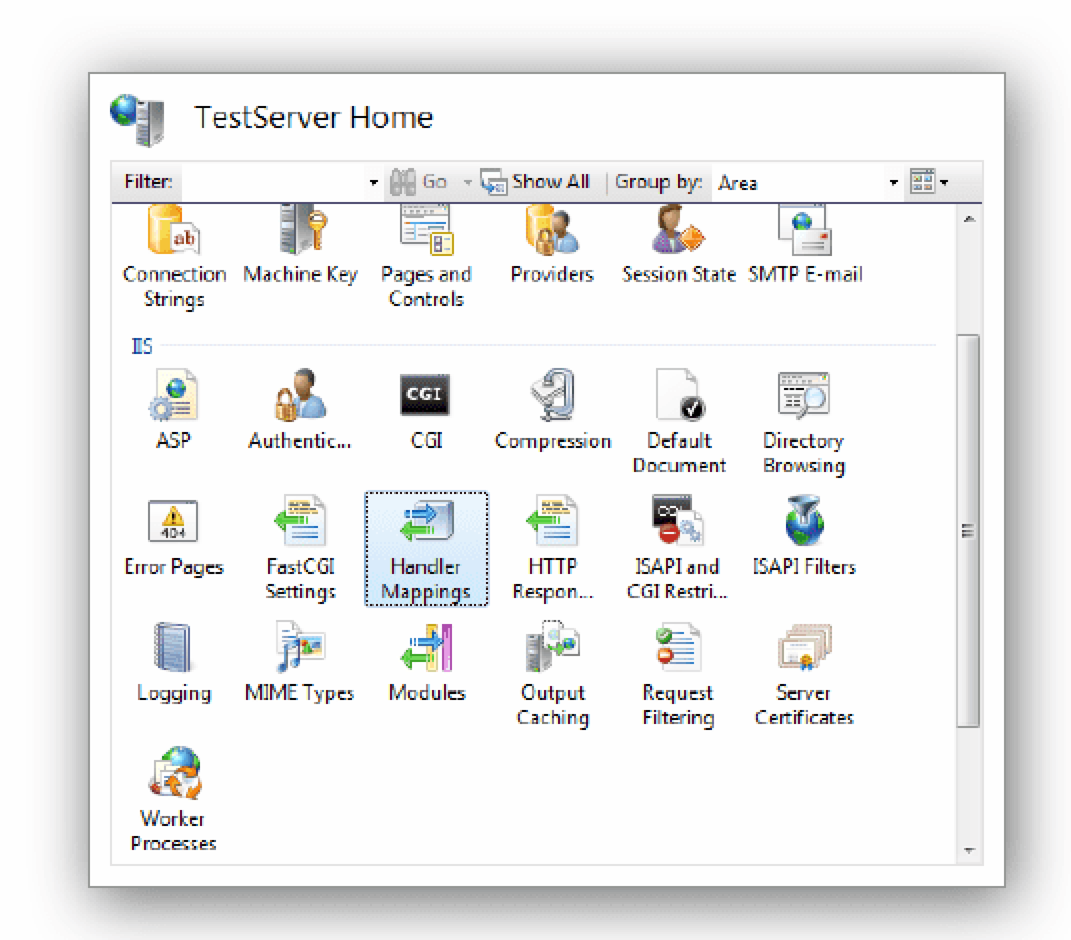


图 2-57　微软 IIS 服务欢迎页面

2.7.2　为微软 IIS 服务添加 PHP 支持

由于微软 IIS 服务对 PHP 不提供原生支持，所以需要为其添加 PHP 支持。首先，需要下载 PHP 引擎包，这一点与 2.3 节中的内容类似，我们就不多说了。将下载解压好的 PHP 引擎包放在一个容易找到的路径下，然后打开“php.ini_recommended”文件，将其重命名为“php.ini”，按照 2.3 节中的方法修改“php.ini”文件。然后将修改好的“php.ini”文件放到“C:\Windows\”路径下。

做好这项工作后，需要打开 IIS 管理器，单击管理器左侧导航树的顶级节点（如图 2-56 所示的“TestServer”节点）。然后在管理器的中部区域框中找到并双击“处理程序映射（Handler Mappings）”图标，进入“处理程序映射”区域框，如图 2-58 所示。



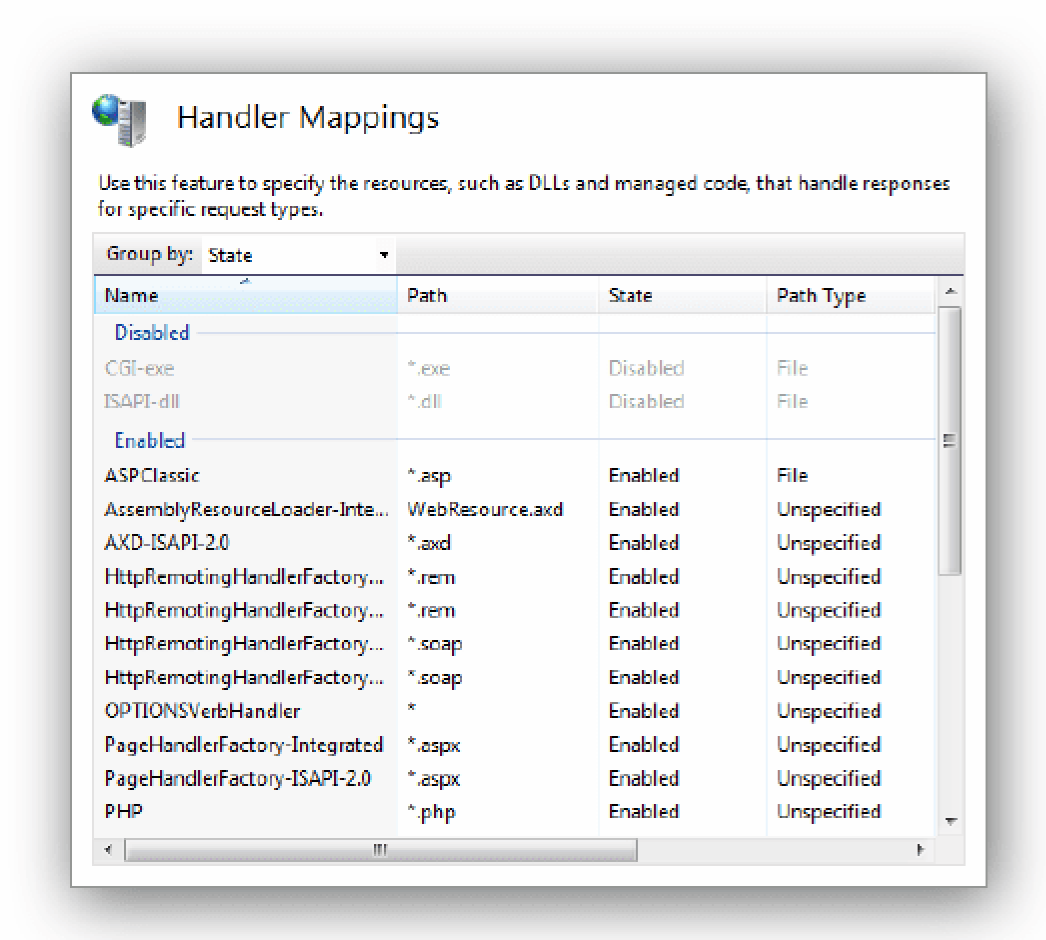


图 2-58　处理程序映射区域框

在“处理程序映射”区域框的空白处，单击右键弹出快捷菜单。在快捷菜单中选择“添加模块映射（Add Module Mapping）”命令，弹出“添加模块映射”对话框，如图 2-59 所示。

按照图 2-59 所示填写好各项参数。需要注意的是，“可执行文件（Executable）”参数需要指定“php_cgi.exe”文件的路径，读者可以在 PHP 引擎包解压后的路径中找到它。填写好各项参数后，单击“确定（OK）”按钮，关闭对话框并回到“处理程序映射”区域框。你会发现，刚才添加的处理程序映射出现在了在区域框中。

接着，还需要为 PHP 应用指定默认文件。

打开 IIS 管理器，选中左侧导航树的顶级节点。然后双击管理员中部区域框中的“默认文件（Default Document）”图标，进入“默认文件”区域框。

在该区域框的空白处单击右键。在弹出的快捷菜单中选择“添加（Add…）”命令，弹出“添加默认文件”对话框。在对话框中输入“index.php”后，单击“确定（OK）”按钮，关闭对话框。

至此，微软 IIS 和 PHP 处理引擎就算是认识了。

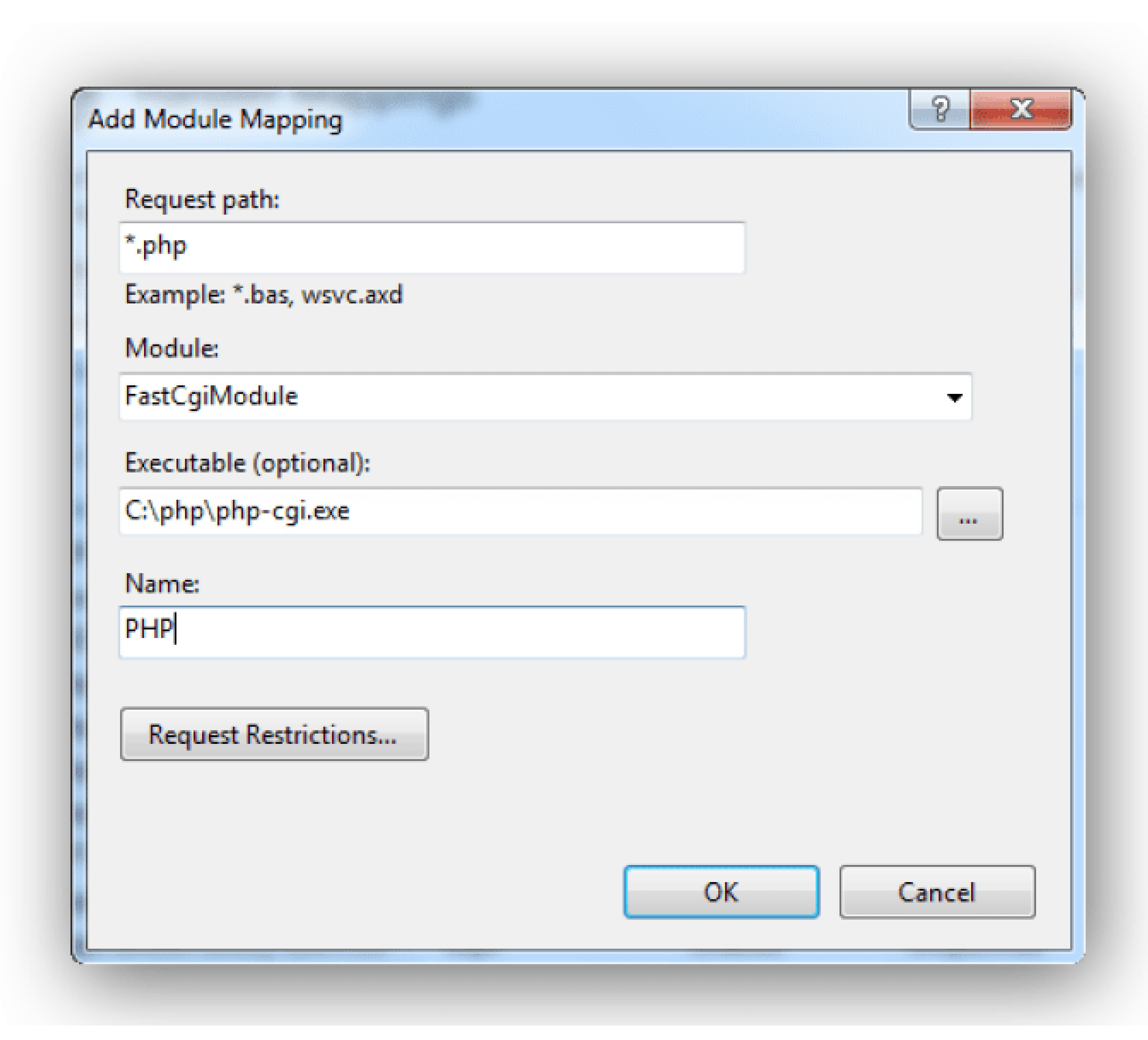


图 2-59　添加模块映射

2.7.3　验证微软 IIS 服务对 PHP 的支持

现在来验证一下微软 IIS 服务对 PHP 的支持情况。首先在左侧导航栏的“站点”节点上右键单击，弹出快捷菜单。在快捷菜单中，选择“新建站点（Add New Site）”命令，弹出“新建站点”对话框，如图 2-60 所示。

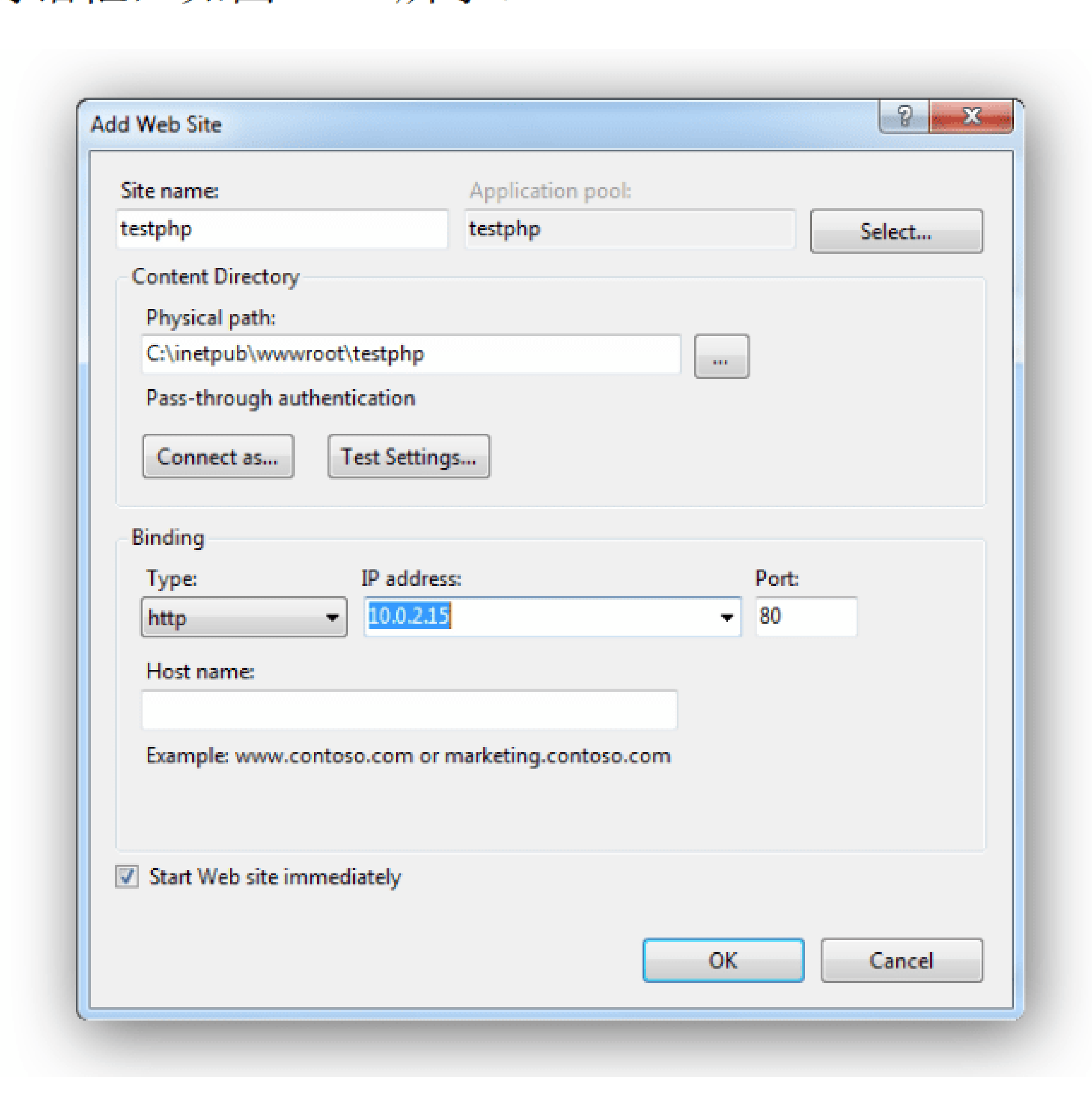


图 2-60　添加新站点

在填写好各项参数后，单击“确定（OK）”按钮，关闭对话框。这时，刚才添加的站点就出现在了左侧导航树的“站点”节点下。

接下来需要在站点的根目录下添加一个名为“index.php”的文件，然后在文件中输入如下代码：

```
<?php
phpinfo();
?>
```

代码输入完毕后，保存该文件。然后回到 IIS 管理器，选中“testphp”节点，然后单击管理器窗口右侧的“浏览站点”区域框中的链接，即可在浏览器中查看该网站了，如图 2-61 所示。

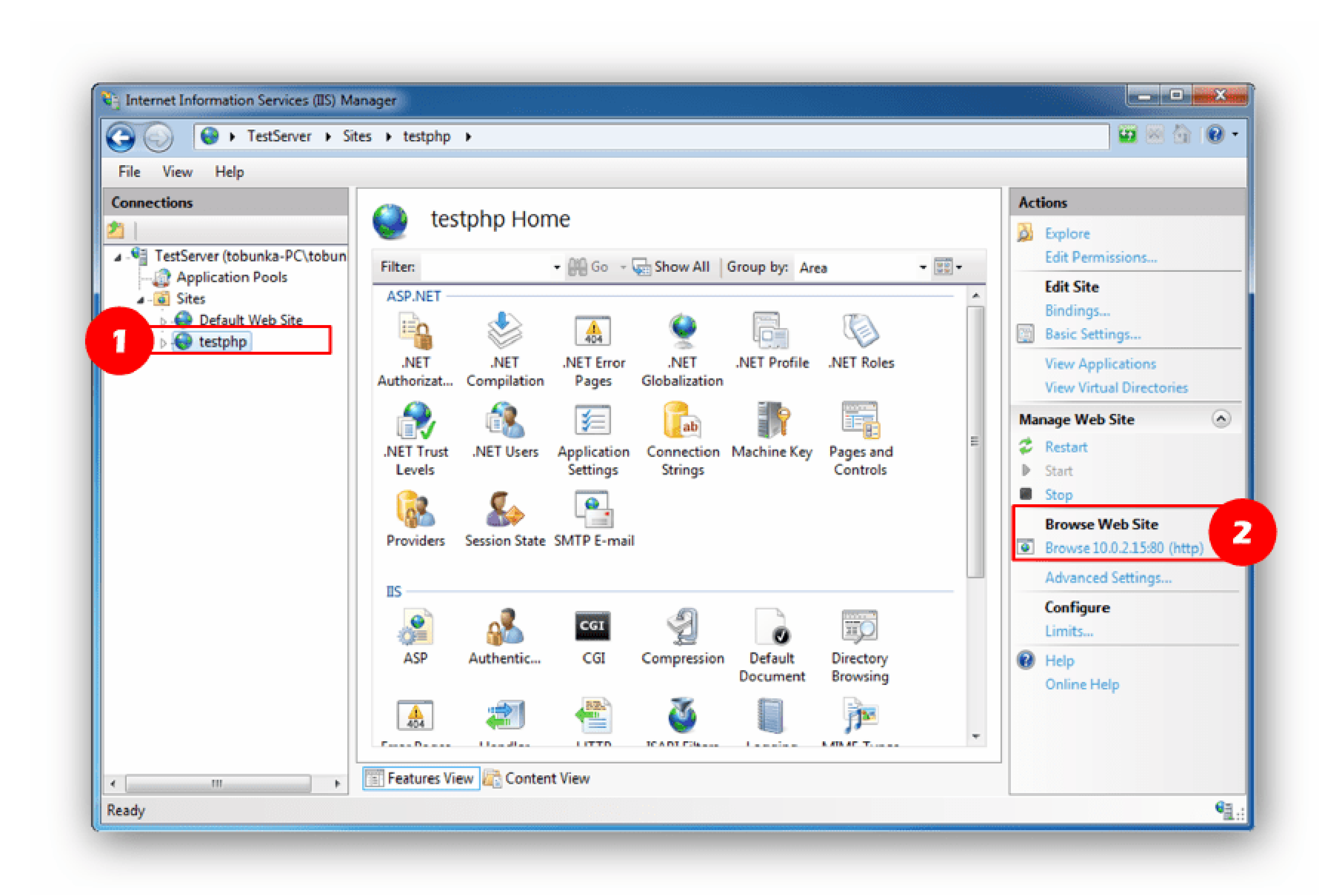


图 2-61　查看“testphp”站点

2.8　安装集成开发环境（IDE）

工欲善其事，必先利其器。在搭建完成 PHP 运行环境后，还需要一个集成开发环境用于编写和调试 PHP 脚本。虽然说用文本编辑器也可以完成 PHP 脚本的开发和调试，但是使用文本编辑器毕竟没有集成开发环境来得快，同时也会增加 PHP 学习曲线的陡峭程度。一旦在使用文本编辑器编写脚本出错调试无果时，很容易使 PHP 新手产生畏难的情绪，导致放弃学习。

说起来，PHP 真的不难，难就难在脚本的调试和维护。为了能够更加聚焦于 PHP 脚本的编写，建议大家还是使用集成的开发环境，不仅上手更快，脚本维护起来也更方便。

2.8.1　IDE 是什么

简而言之，IDE 为编码提供一站式服务的环境。一个完整的 IDE 通常包括一个编辑器

用于编辑和调试代码，还包括一个内嵌的浏览器用于查看脚本的运行情况。有的 IDE 还集成了版本控制功能。

一般来说，IDE 具有如下几个特点。

1．基于项目的开发

IDE 的一个关键特性是它把一个 PHP 应用程序看作是一个项目，而不仅仅是一组文件。在这个概念下，IDE 可以帮助我们更好地控制源码、调试数据库，以及规划某一关键目录的所在位置。

2．调试

IDE 的另一个方便的特性是集成调试。这样一来，可以在编辑器中设置断点，当 PHP 解释程序执行到这个脚本时就会停止，以便检查局部变量的值，从而诊断脚本中出现的问题。当然，也可以在脚本中可以使用 echo 语句来检查值或者也可以使用错误日志获得变量的值。

3．代码智能补全

PHP 是一种非常规则的编程语言，这意味着它遵循着简单的模式。这些模式不仅使代码易于编写，也使 IDE 在项目中检查代码变得很容易。此外，这一功能还可以帮助我们更快更准确地编写脚本。举个例子，如果我们在项目中定义了一个名为 MyClass 的类，那么当我们在编辑器中输入了关键词 new 之后，光标附近立即会出现一个包括 MyClass 作为选项的列表。无论使用何种类型的对象，IDE 都会立刻显示它的可用方法和实例变量。这应该是使用 IDE 而不是文本编辑器的首要原因。这种代码智能补全功能可以有效减少敲错类名、方法名和参数。

4．类视图

IDE 中的代码智能引擎产生的另一个作用是 IDE 可以产生项目的类视图。使用这一功能，可以方便地了解当前项目中已经定义的所有类，而不用去担心找不到定义这些类的文件。当使用这些类时，可以方便地通过编辑器访问相应文件并显示相应类、方法或者实例变量。

5．多语言支持

大多数的 IDE 不仅支持 PHP 而且支持如 JavaScript、Structured Query Language（SQL）、HyperText Markup Language（HTML）和 Cascading Style Sheets（CSS）等相关的语言集。因为 HTML 和 CSS 比较简单，所以 IDE 对它们的支持是最好的。对于 JavaScript 的支持通常会有兼容性问题也导致代码智能补全失效，但是支持比不支持要好。

6．源码控制

大多数的 IDE 都支持一些与源码控制系统的连接，以方便随着时间的推移维护项目文件的不同版本，并可以标记某一特定的版本为发布版本。在团队环境中使用源码控制系统

是十分重要的，但这并不是说源码控制对于个人使用不重要。当磁盘瘫痪或者客户突然想要以前的版本而不是现在的版本的时候，一个好的源码控制系统就可以发挥作用了。

7．FTP/SFTP 集成

与源码控制相关的一种功能是在服务器中对于最新的代码使用 FTP。这比使用 FTP 客户机或者自己打包文件并发送给服务器然后再解包要容易许多。

8．数据库导航

有些 IDE 会提供数据库导航功能。使用这个特性，可以浏览应用程序访问的数据库、找到表格和字段名并返回查询结果。一些系统甚至可以自动写入一些数据库访问代码。

9．集成Web浏览器

一些 IDE 支持集成 Web 浏览器，可以直接导航到正在使用指定的附加参数编辑的页面。既可以直接使用内置在 IDE 中的，也可以调用外部浏览器。建议大家还是多使用外部浏览器，因为内置浏览器对源码的解析可能会与用户使用的实际情况有差异，从而导致页面解析出现的效果并不是想要的样子。

10．片段

基本上所有的 IDE 都提供了方便快捷地保存代码片段的功能。所谓片段，其实就是完成小任务（比如在一些输入中运行常规表达式、连接到数据库和查询数据库）的小部分代码。读者可以将经常使用的一些代码片段保存起来，以方便随时调用。

2.8.2　PHP 开发中常用的 IDE

在 PHP 的开发中，比较常见的 IDE 有如下几种。

1．PHPEclipse

PHPEclipse 是基于 Eclipse 独立开发的一套 IDE。可以方便地运行在 Windows、Linux 和 Unix 平台上。支持 2.8.1 小节里提到的几乎所有的功能。是一款专注 PHP 项目开发的 IDE。

2．Komodo IDE/Edit

Komodo IDE 是一款商业软件，除了支持 PHP 开发之外，还支持 Python、Ruby 和 Perl 等语言的开发。功能十分的强大。如果不想付费，可以使用 Komodo Edit。这是一款免费的脚本编辑器，集成了 Komodo IDE 的代码智能补全功能。Komodo IDE 令人称道的一点是，它集成了一个正则表达式调试器，可以方便地调试正则表达式。

3．Aptana Studio/Plugin

Aptana Studio 是一款基于 Eclipse 的开源的 IDE。与 Komodo 类似的是，它支持多种编程语言，可以很方便地同时管理多个不同语言的项目，同时它的代码智能补全系统十分的

健壮，对于提高代码输入效率和代码准确性来说，不可多得。另外，它还为已经熟悉 Eclipse 开发流程的人准备了 Aptana Plugin。如果不想让电脑上的 IDE 成堆，那么也可以试试它的 Plugin，功能与 Studio 版本是一样的。

4．Zend Studio

Zend Studio 是一款号称专业的 PHP 项目开发必不可少的 IDE。不过，它是一款商业软件，使用它可是要付费的。与 PHPEclipse 类似的是，它也专注于 PHP 开发，上面提到的各种功能它都支持。如果是企业级应用的开发，建议还是用专业的 IDE。不过对于看这本书的读者来说，就不是那么必要了。

5．PHPEdit

从严格意义上来说，PHPEdit 不能算是 IDE。不过它也有强大的代码智能引擎，可以提高代码输入的效率和准确性。因此，姑且把它摆在这里。

如果将上面这五种 IDE 按功能分类的话，可以分为两类，一类仅支持 PHP 项目的开发，如 Zend Stuido、PHPEclipse 和 PHPEdit；另一类则支持多种语言项目的开发，如 Komodo IDE 和 Aptana Studio。大家可以根据自己的需要取舍一下。

2.8.3　创建 PHP 项目

在这一小节里，我们选择 Aptana Studio 来创建第一个 PHP 项目。选择 Aptana Studio 的原因是因为它强大的代码智能引擎，可以极大地提升脚本编写效率和准确率。同时，其项目化的管理，也可以帮助大家很快地上手 PHP 编程。

现在 Aptana Studio 主推的版本是 Aptana Studio 3。大家可以访问它的官方网站获取。按照官方网站的说法，Aptana Studio 3 有六个核心功能。它们分别是：代码助手、部署向导、集成调试、版本控制集成、内置终端和 IDE 自定义，如图 2-62 所示。

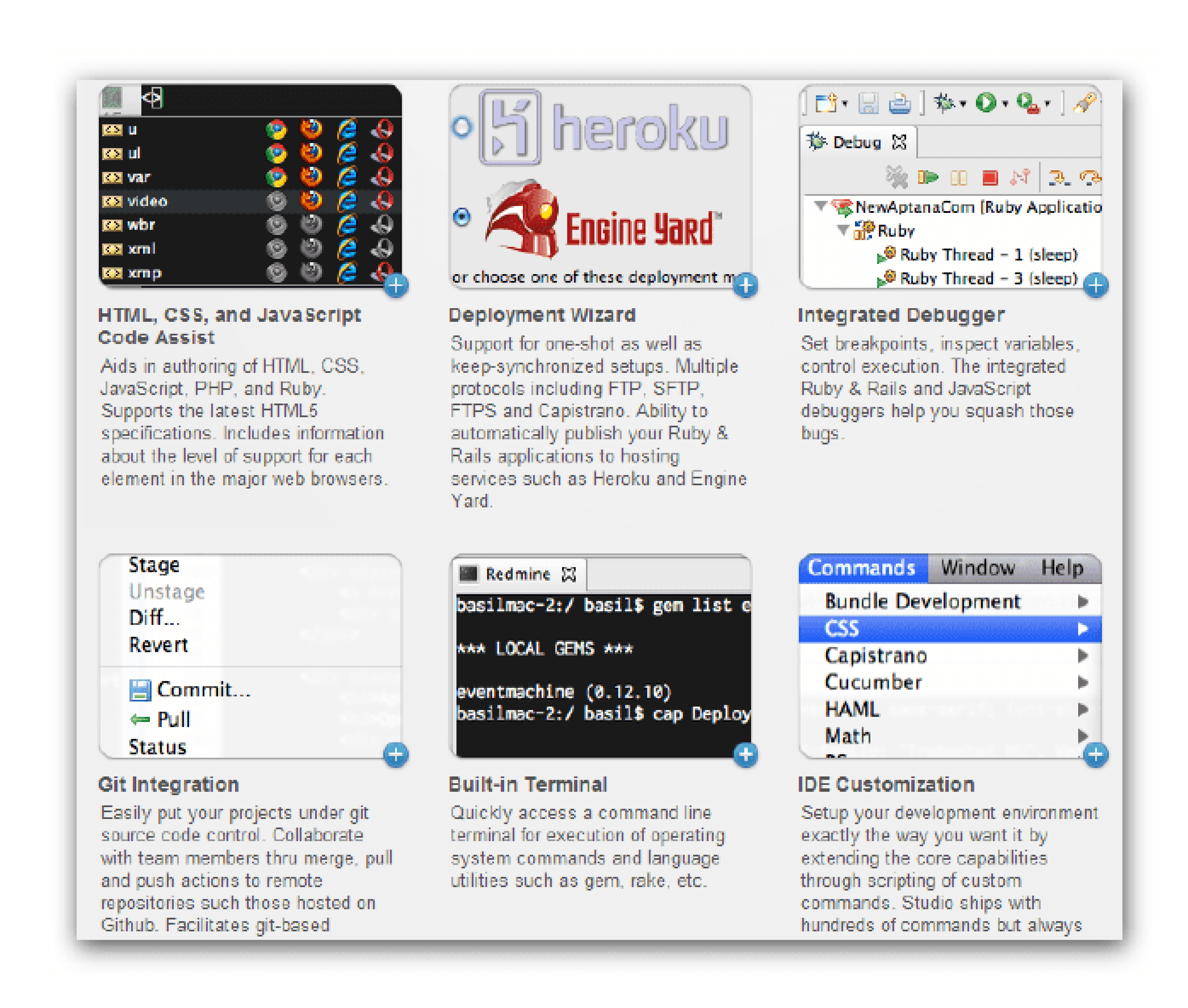


图 2-62　Aptana Studio 3 核心功能介绍

Aptana Studio 3 在 Windows 操作系统中的安装过程与其他应用程序没有太大的区别。在安装过程中，建议都使用默认的配置，直接单击“下一步”按钮就好。

在安装结束后，第一次打开 Aptana Studio 的时候，会弹出一个对话框让我们指定默认的工作区（workspace）。当指定了工作区后，就可以看见 Aptana Studio 的开始页面了（如图 2-63 所示）。在这个页面中，可以接入 Aptana 的讨论区、帮助文档和 Bug 库。

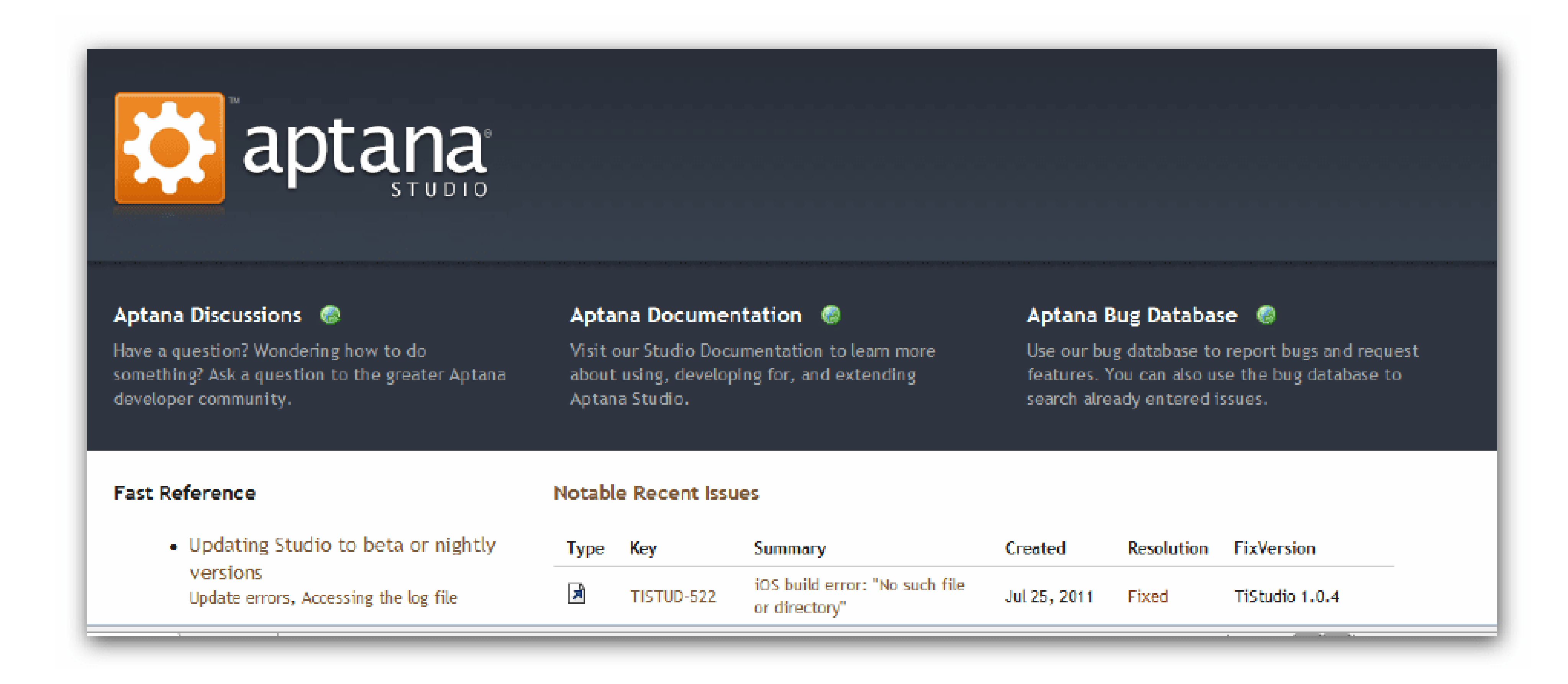


图 2-63　Aptana Studio 开始页面

现在，先来认识一下 Aptana Studio 的界面，如图 2-64 所示。

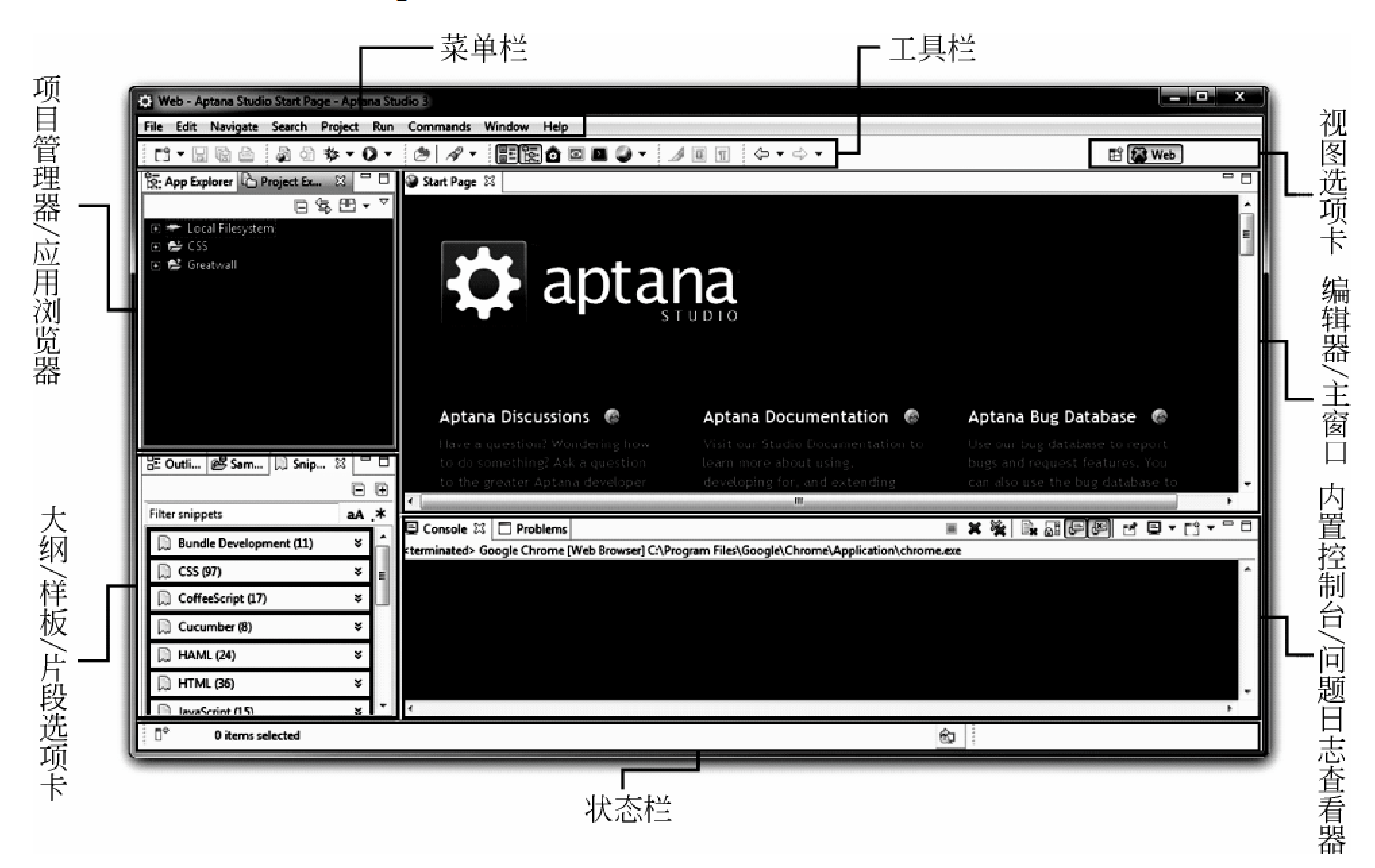


图 2-64　Aptana Studio 界面截图

在安装好 Aptana Studio 并设置好默认工作区之后，就可以创建 PHP 项目了。选择 File |New… | PHP Project 命令，弹出如图 2-65 所示的窗口。

在窗口出输入项目名称和使用的 PHP 版本，然后单击窗口下方的 Finish 按钮就算成功创建了一个 PHP 项目。在新创建的 PHP 项目中默认是没有任何文件的，读者可以在 PHP 项目创建 PHP、HTML、CSS 和 JS 等一切与项目有关的文件。

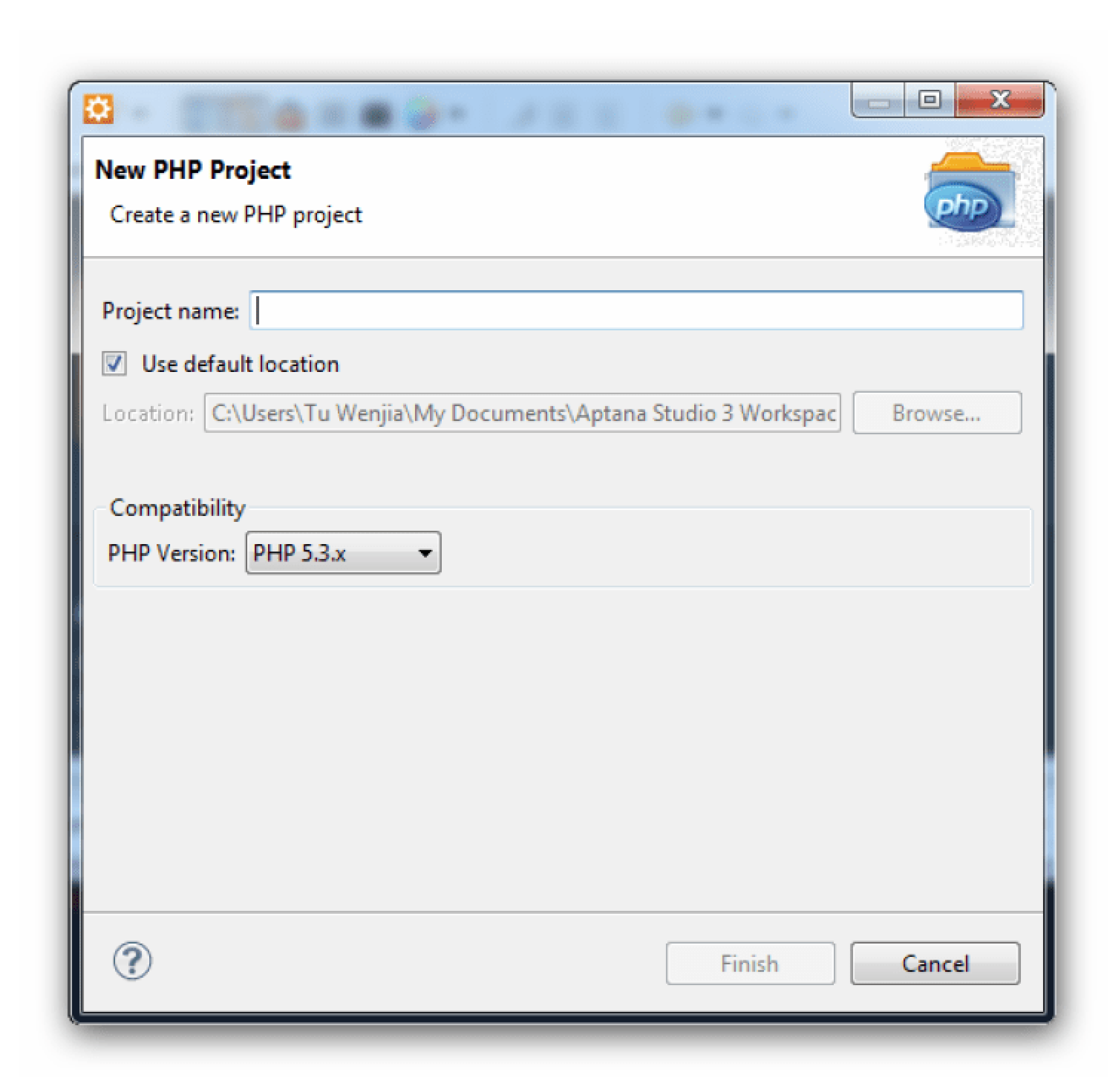


图 2-65　创建 PHP 项目

2.9　习　　题

（1）如果读者使用的是 Microsoft Windows 系列的操作系统，请使用 Apache 2.x、PHP 5.2 和 MySQL 5.x 搭建 PHP 的运行环境。

（2）如果读者使用的是 Microsoft Windows 操作系统中自带的 IIS 服务器，请按照 2.7 节中的描述使用 PHP 5.4 和 MySQL 5.x 来搭建 PHP 的运行环境。

（3）请按照 2.8 节中的描述安装好集成开发环境。

第 3 章　动手写第一个 PHP 脚本

所谓 PHP 脚本，其实就是一串指令，告诉 PHP 处理引擎应该完成什么动作。理论上来说，PHP 脚本可以只包含一条命令，也可以包含成千上万条命令，这完全取决于读者的需要。对于一个 PHP 脚本来说，PHP 处理引擎是按照从上到下、从左到右的顺序一条一条处理的，直到引擎指针指向脚本的最后一行命令。

那么，我们可以用 PHP 脚本做些什么事情呢？按照本书 1.2 节中的说法，我们可以编写 PHP 脚本实现在特定的网页显示特定的内容、将用户在表单中填写的内容存入数据库、将某目录中的文件备份到指定的存储设备上。PHP 几乎无所不能，只要读者肯下功夫，学习 PHP 是一件十分简单并且充满乐趣的事。

在本章里，我们将要动手写下第一个 PHP 脚本。

3.1　何谓 PHP 命令

按照表现形式的不同，PHP 命令可以分为简单命令和复杂命令两种。如何判断一条命令是简单还是复杂呢？

3.1.1　简单命令

每条简单的 PHP 命令都在告诉 PHP 处理引擎执行一个动作。最常见的 PHP 命令就是 echo 命令，它的功能是显示和输出信息。在第 1 章里，我们就已经见过这条命令了。

现在再来详细地看一下这个命令。

【例 3.1】 echo 命令。

```
echo "Hi";
```

在这条命令中，有三个部分组成。它们分别是命令关键字 echo、命令对象 Hi 和行结束符（;）。当 PHP 处理引擎读到这条命令时，它首先会看到这条命令的关键字，通过关键字了解命令要求完成的动作；然后再读取命令的对象，并按照命令关键字的要求完成对对象的处理；最后引擎会读取行结束符来结束对这条命令的执行。

刚才说到，echo 命令的功能是显示和输出信息，那么当 PHP 处理引擎读到这条命令时，就会输出一个简单的字符串“Hi”。

这个例子十分好懂，也没有什么好讲的。不过有的同学可能对行结束符产生了兴趣。行结束符和平常 Word 文档里的回车符有什么本质上的区别么？为了讲清楚这个问题，再来看几条命令。

【例 3.2】 行结束符。

```
1  echo "Great!
2  I hope I can finally get there!";
3
4  echo "Great!"; echo "Well done!";
```

看到这里，有的同学可能就不淡定了：这到底算是三条命令还是两条命令呢？嗯，这个问题问得好。其实答案也很简单，那就是只有当行结束符出现的时候，一条命令才算结束，无论一条命令被切成了几段写在了几行里。同理，若干条简单命令只有行结束符齐全，也可以挤在一行里抱团取暖，就像例 3.2 中的第 4 行一样。即便如此，还是建议大家一行只写一条简单命令，这样在后期进行错误定位的时候会好过得多。

PHP 引擎其实无从知晓代码的内容，它只知道寻找行结束符。在两个行结束符之间的内容就会被 PHP 引擎当成一条命令加以执行。于是上面这条命令的结果就是另一个简单的字符串“Great! I hope I can finally get there!”。

好奇的同学可能又要问了：如果在一个脚本里一个行结束符都没有的话，是不是所有的代码就会一起执行呢？答案是肯定的，但是你却看不到你想要的结果，看到的只是如下的一条报错信息：

```
Parse error: expecting "," or ";" in file.php on line 6
```

在报错信息中，你会看到出错的文件名和具体的行号，以及可能解决问题的办法。通常情况下，在每一条命令结尾处加上一个分号就能解决这个问题。

对于一个只有几条命令组成的 PHP 脚本文件来说，定位错误是十分容易的一件事情。但是通常情况下，一个 PHP 脚本怎么着也得有个上百条命令。因此选用一款可以显示行号的编辑器就成了一个明智的选择。否则，你就只能从上往下一条一条地数了。

3.1.2　复杂命令

把若干条简单命令放到一对花括号里，这些命令就组成了一个复杂命令。一个复杂命令通常包含若干条简单命令，甚至还会嵌套一些复杂命令。最常见的复杂命令块就是条件命令，只有满足特定条件时，花括号中的简单命令才会被执行，如例 3.3 所示。

【例 3.3】 If 条件命令。

```
if (time is in the morning)
{
  get up;
  brush my teeth;
  wash my face;
  put on my jacket;
  go to work;
}
```

在上面这个例子里只出现了一对花括号。这对花括号中包含了 5 条简单命令。这个例子可以做如下的解读：

```
早晨，我起床后会先刷牙，再洗脸，然后穿上我的夹克出门上班。
```

在这句简单的描述中，我们可以发现作为条件的时间是早晨。也就是说只有早晨，我

才会做如下的动作：起床、刷牙、洗脸、穿衣和出门上班。于是，需要把条件写在花括号外的 if 子句中，然后把当条件满足时需要完成的动作依次写在花括号内，从而完成一个复杂的条件命令。

对于一条复杂命令来说，PHP 会一次性读取这条复杂命令所有的内容。值得注意的是，花括号后面是不需要加行结束符（;）的。

另外，大家还要注意务必让花括号内的每条语句都缩进若干字符并使它们保持左对齐。这个要求并不是必须的。但是，如果你和其他的同事都在编辑同一个 PHP 脚本，为了他人阅读的方便还是建议大家照做。

3.2　如何写代码

第 1 章里提到动态网页这个概念。为了使网页“动”起来，就得在 HTML 代码中插入 PHP 脚本，然后将这些内嵌 PHP 脚本的 HTML 文件保存为扩展名为.php 的文件。如此一来，PHP 处理引擎才会处理文件中的 PHP 脚本。

本节将讨论一个 PHP 脚本应该包括的元素。

3.2.1　PHP 标记对

只有当 PHP 脚本被插入扩展名为.php 的 HTML 文件中时，PHP 引擎才会处理这些脚本。那么应该把这些脚本插入到 HTML 文件中的什么地方呢？先来看例 3.4 中的代码。

【例 3.4】 PHP 标记对。

```
<?php
…
PHP statements
…
?>
```

所有的 PHP 脚本都应该被包含在如例 3.4 所示的标记对中。读者也可以使用“<?”和“?>”来标记一个 PHP 脚本的起止。前提是修改了 php.ini 文件中关于启用短标记对的相关内容。

一般来说，使用短标记并不是一个好主意。如果把使用短标记对的 HTML 文件转移到一台没有启用短标记对的服务器上，那么所有的 PHP 脚本都会失效。尤其是对于租用服务器的开发者来说，这样做的后果是致命的，因为大多数供应商并不允许修改 php.ini 文件。这样一来，编码时少敲几个字母的代价也忒大了些。所以还是建议大家尽量使用完整的 PHP 标记对。

第 1 章的结尾曾经提到：如果用户通过浏览器发出访问请求，PHP 标记对间的所有 PHP 脚本都会被送到 PHP 处理引擎进行处理。然后服务器将经过处理的页面下发到发出请求的浏览器。该页面中所有的 PHP 脚本都已经被替换成了相应的处理结果。在浏览器里通过查看源代码的方式是无法看到任何 PHP 脚本的。

例如，读者可以在 HTML 代码中加入如例 3.5 所示的 PHP 脚本，然后将 HTML 代码

保存为一个 PHP 文件。

【例 3.5】 PHP 脚本。

```
<?php
    echo "This line is brought to you by PHP.";
?>
```

当用户请求该页面时，服务器会先查看文件的扩展名。当服务器发现该文件是一个 PHP 文件时，安装在服务器上的 PHP 处理引擎就会检查该文件里的 PHP 标记对、执行标记对中的脚本、并输出相应的结果。在本例中，服务器上的 PHP 处理引擎会执行 PHP 标记对中的 echo 命令，并输出处理结果，也就是"This line is brought to you by PHP"这句话。

当执行完文件中应该执行的所有脚本后，服务器会用脚本的执行结果替换相应的脚本，然后将处理后的 HTML 文件下发到用户的浏览器中。用户就能看到上面那句话了。

3.2.2　注释脚本

看到这一节的标题，有的同学会问：为什么要注释脚本呢？

注释对于脚本来说十分重要。通常情况下，我们会使用注释来描述代码，告诉阅读脚本的人某一段代码可以实现的功能以及该功能是如何实现的。当脚本十分复杂，让人无法很快读懂时，注释就显得尤为重要了。但是如果代码只有自己一个人在维护，那么是不是就不用注释了？自己写的代码难道自己还看不懂吗？对于这个问题，我只能用一句俗语来回答："好记性不如烂笔头。"既然我们可以很方便地在脚本旁边注明一下某段脚本的功能，为什么不呢。更何况，脚本会变得越来越复杂，总有一天会需要很多的人来一起维护。写上注释就可以避免出现代码无法维护的情况，提高代码的利用效率。

所谓注释，其实就是写在脚本旁边用于说明代码的一段文字。PHP 处理引擎在碰到注释时会直接忽略。也就是说，注释一定是给人看的，那么写注释的时候言简意赅就显得十分必要了。那么 PHP 处理引擎如何区别脚本和注释呢？还是通过一个例子来说明一下。

【例 3.6】 注释示例。

```
/*   在这儿写注释
在这儿写更多的注释    */
```

在例 3.6 中，我们看到了如下的两个符号："/*"和"*/"，这样的注释标记称为长注释标记。PHP 处理引擎在看到这一对符号时，就会直接忽略它们之间的所有内容。大家可以在开始写脚本之前，在开头的地方注释一段，写一写脚本的名字、描述、作者信息和写作时间等信息，以后查找起来也会非常方便。例 3.7 就是一段脚本说明。

【例 3.7】 脚本说明。

```
/*  name: hello.php
    description: Displays "Hello World!" on a web page.
    written by: Joe Programmer
    created on: Feb 1st, 2012
    modified on: Mar 15th, 2012
*/
```

值得注意的是，长注释标记不支持嵌套。也就是说，如果出现了如例 3.8 这样的注释

标记，PHP 会报错。

【例 3.8】 错误的注释嵌套。

```
1   /* 这是一条注释
2     /* 这是另一条注释 */
3   */
```

有人可以会问了，这不是挺工整的吗，为什么会报错呢？我们来分析一下：按照之前的说法，PHP 处理引擎在见到“/*”符号时，就会忽略之后的所有内容，直到它遇到了“*/”符号。这样看来，在例 3.8 中，PHP 处理引擎会把第一行的“/*”和第二行的“*/”当成是注释的开始和结尾，而把第二行开头的“/*”当成了注释的一部分。那么第三行的“*/”就形只影单无人顾了。PHP 处理引擎也会因为无法处理这个形只影单的标识符而报错。

对于注释内容如例 3.7 这样比较多的情况来说，这个注释标记还显得不是很累赘。如果注释只有一行，还要陪上 4 个字符的注释标记对，效率实在是太低了。其实 PHP 还提供了两种短注释，标识符是井号（#）或双斜杠（//），如例 3.9 所示。

【例 3.9】 短注释。

```
# This is a comment.
//This is another comment.
```

那么这两种标识符有什么区别呢？井号（#）只能用在一行的开始，而双斜杠（//）可以用在一行的中间。当需要在某一行命令后进行注释时，可以使用双斜杠（//），如例 3.10 所示。

【例 3.10】 双斜杠注释符可以用在一行的末尾。

```
<?php
    echo "Hello World!";    //打印“Hello World!”
?>
```

在本书中，这三种注释标识都会用到。为了统一风格，也为了帮助大家养成注释脚本的习惯。本书做出如下的规定：

- ❑ 脚本的开始使用长注释书写脚本说明。
- ❑ 若一个脚本中包含有若干个模块，在每个模块开始前用短注释标识（#）说明模块的功能。
- ❑ 在重要的命令行后用短注释标识（//）说明该命令行的作用。

3.3　实战练习：向世界说 Hello!

虽然之前就已经提到过 echo 命令的功能和用法，但是并没有形成一个系统的概念。为了更好地使用这条十分常用的命令，很有必要在正式地用 PHP 脚本向世界说你好之前系统地讲解一下这条命令。

3.3.1　echo 命令初识

这条命令可以说是编写 PHP 脚本必用的命令之一。没有哪个脚本在被执行之后不输出信息的。一旦需要输出信息，就一定会用到这条命令。例如，编写了一段在操作系统里查

找某个特定文件的脚本。查找的过程当然是不可见的，但是如果连查找的结果也不输出的话，如何知道这个脚本是不是起作用了，要查找的文件是不是已经找到了呢？对于这样的脚本，通常需要输出的信息包括，查找结果和需要查找文件的文件名等。如果在操作系统中查找到了该文件、则还需要输出该文件的存储路径、存储时间、最后修改时间和摘要信息。这样一来，我们才能知道编写的脚本是不是起了作用、是否找到了需要的文件以及文件的基本情况。

所以说，echo 命令是十分重要也是十分必要的一条命令。按照之前的示例中书写的样式，我们可以总结出 echo 命令的基本样式。

【例 3.11】 echo 命令的基本样式。

```
echo output1, output2, output3, …
```

在使用 echo 命令时，一定要注意以下几点：

- 输出的对象，也就是例 3.11 中的 output 参数一定是一个数字或者字符串。所谓数字就是像 1 或者 234 这样的数字（如 echo 93;），而字符串则是一串包含在引号内的字符（如 echo "Hello World!";）。关于引号的使用，将在第 7 章讲字符串型变量的时候会进行详细地讲解。
- 理论上来讲，一条 echo 命令可以输出的对象是无限的，但是当输出对象多于两个时，需要在相邻的输出对象之间加上一个逗号。千万注意，不要画蛇添足的在逗号后面加个空格，否则脚本会报错。
- 空格也是一种字符，可以通过在空格前后添加引号来输出（如 echo "　　";）。

表 3-1 所示列出了使用一些 echo 命令和它们的输出结果。

表 3-1　echo命令的输出结果

echo 命令	输 出 结 果
echo 159;	159
echo "Hello World!";	Hello World!
echo "Hello","World!";	HelloWorld!
echo "Hello"," ","World!";	Hello World!

在使用 echo 命令时，还要注意一些诸如“\n”、“\t”和“\”这样的特殊字符和转义字符。具体的内容会在第 7 章讲解字符串类型的变量时进行详细地讲解。

到这里，使用 echo 命令需要注意的相关内容都呈现给大家了。是不是很简单呢？有没有学会靠事实来说话。下面就让我们进入到实战练习吧。

3.3.2　实战练习——向世界说 Hello!

在本小节里，我们将使用 PHP 基础知识来写一段脚本，用不同的语言向世界说 Hello!

在示例脚本中，会出现一些之前没有见过的内容，比如说数组、再比如说循环等等。这些都是 PHP 的基础知识，大家会在后续的章节中学到。另外，脚本中涉及到关于 HTML 和 JavaScript 的相关知识，书中会附带着给予相应的解释，应该不会给大家的阅读带来影响。不过，这里也有一个小建议，那就是尽可能多的掌握 HTML 和 JavaScript，特别是在 HTML 5 大行其道的年代，这一点非常的重要。

好了，言归正传。首先，先来新建一个名为 01_hello_world.php 的文件，然后在文件中写下一些 HTML 代码：

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Hello World!</title>
        <style>
            #msg {
                font-size:500%;
                margin-left:60px;
                width:900px;
                height:300px;
                line-height:300px;
                text-align:center;
                overflow:hidden;
            }

            #lang {
                font-size:150%;
                margin-left:120px;
                height:80px;
                line-height:80px;
                text-align:right;
            }
        </style>
    </head>
    <body>
        <div id="msg"></div>                    //输出问候的位置
        <div id="lang"></div>                   //输出该问候对应语言的位置
    </body>                                     //在该标签前插入 PHP 脚本
</html>
```

在上面这份 HTML 文件中，定义了两个 div 标签，分别用来存放问候语和该问候语对应的语言。

接下来，需要在“**</body>**”标签前插入如下的 PHP 脚本：

```
<?php
    $msgArray = array('Hello World!',
                 '世界，您好！',
                 '世界，您好！',
                 'こんにちは，世界！',
                 '안녕하세요!',
                 'Bonjour le monde!',
                 'Hallo Welt!',
                 'ສະບາຍດີໂລກ',
                 'saluton mondo');  //定义了一个数组，用于存放问候语

    $langArray = array('English',
                  '简体中文',
                  '繁体中文',
                  '日本語',
                  '한국어',
                  'française',
                  'Deutsch',
```

```
                    'ພາສາລາວ',
                    'Esperanto'); //定义了另一个数组，用于存放问候语对应的语言
    $idPrev = 0;

    /*因为在下面的循环里使用了 rand()产生的随机数，为了避免重复出现两句相同的问候
    这里定义了一个比较变量*/

    for ($i=0;$i<=10;$i++){          //开启一个循环，用于随机输出问候语
        $id = rand(0,8);             //获取一个 8 以下的随机数
        if($idPrev === $id) {        //判断当前循环中产生的随机数与上一次循环中产
                                       生的循环次数是否相同
            $id = $idPrev + 1;
            if($id > 8)
            {
                $id = 8;
            }
        }                            //<=中产生的随机数是否相同
        $msg = $msgArray[$id];       //获取产生的随机数对应的问候语
        $lang = $langArray[$id];     //获取产生的随机数对应的问候语使用的语言
        $idPrev = $id;               //为下一次循环比较随机数做准备
        echo '<script
language="javascript">document.getElementById("msg").innerHTML="'.$msg.
'"</script>';
        echo '<script
language="javascript">document.getElementById("lang").innerHTML="'.$lan
g.'"</script>';

        /*上面使用学习到的 echo 命令输出了两个字符串。在这两个字符串中我们使用了
          JavaScript 脚本。这个脚本的向 HTML 文件中的 ID 为 msg 的 DIV 标签输出
        变量$msg 的内容，同时向 ID 为 lang 的 DIV 标签输出变量$lang 的内容*/

        echo str_repeat(' ',1024*64);   //重复输出空格以填满缓存
        flush();                        //刷新缓存
        sleep(1);                       //停顿一秒钟
    }
?>
```

当运行上面这段脚本后，会发现浏览器中出现了在数组中定义的问候语及其使用的语言，如图 3-1 所示。

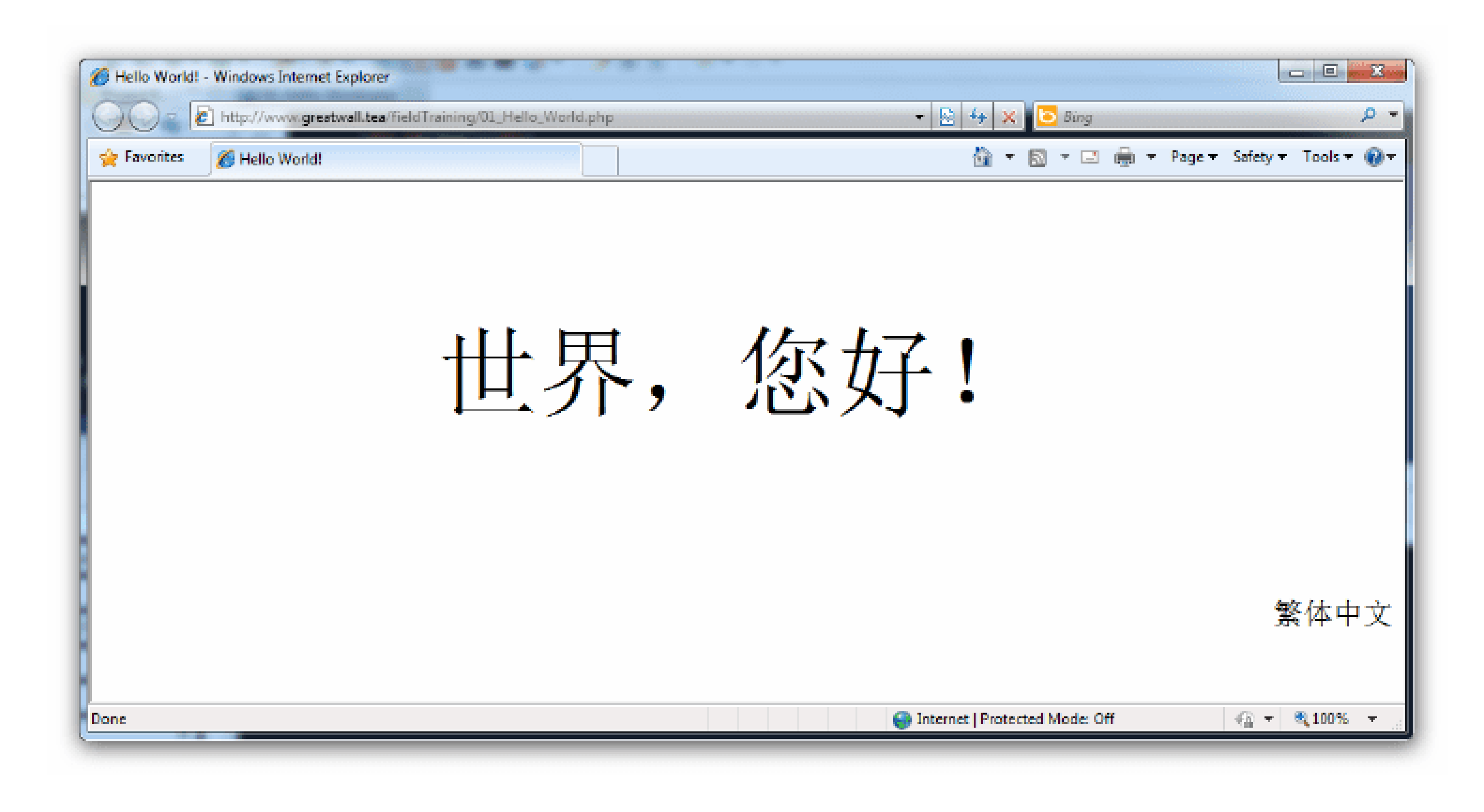


图 3-1　Hello World（繁体中文）

这是怎么做到的呢？一起来看看代码：

在上面这段脚本中，首先定义了两个数组，一个用于存放问候语（$msgArray），另一个则用于存放语言（$langArray）。注意，这两个数组中的问候语和语言是一一对应的，以便于后续从数组中取值时得到的结果也是一一对应的。

由于这些问候语不是按照它们在数组中的顺序在浏览器中出现的。为了防止同一问候语连续多次出现的情况，我们定义了一个变量，用于存放当前问候语对应的数组索引。该变量会在随机产生下一句问候语的数组索引时使用。即$idPrev。这个变量用来存储上一次显示的问候语在数组中的索引。如果当前问候语在数组中的索引与该变量的值相同，则该变量的值加 1，并将加 1 后的结果赋值给存储当前索引的变量。

接下来，我们定义了一个 for 循环用来输出问候语到浏览器。这个 for 循环由四步构成，分别是：

（1）随机生成索引并比较与上一次显示的问候语在数组中的索引。

（2）根据生成的索引在两个数组中查找相应的问候语及其使用的语言。

（3）将查找到的问候语及其使用的语言输出到浏览器。

（4）让系统在刷新缓存后停顿 1 秒钟，再次执行 for 循环。

第一步，定义一个变量$id，用于存储使用 rand()函数生成的一个 8 以内的随机数。若变量$id 的值与$idPrev 相同，则将变量$idPrev 的值加 1 并将计算结果赋值给变量$id，若此时变量$id 的值又大于 8，则将变量$id 的值定为 8。这样一来，就不会出现查找不到问候语的情况了。

第二步，根据生成的索引在两个数组中查找相应的问候语及其使用的语言。具体的做法是，先定义两个变量$msg 和$lang，然后使用在第一步中生成的变量$id 的值做为索引，在数组$msgArray 和$langArray 中查找相应的问候语和使用的语言，并将查找到的内容赋值给变量$msg 和$lang。

第三步，使用 echo 语句输出两条 JavaScript 语句，用来向 HTML 代码中的两个 DIV 标签添加相应的内容。这时，浏览器中就出现了一条问候语及该问候语使用的语言，如图 3-1 所示。

第四步，使用 flush()和 sleep()两个函数用来刷新系统缓存和让服务器暂时休息。关于这两个函数的具体内容，大家可以查看 PHP 官网上的介绍。

至此，一次循环就完成了。浏览器中的信息会每隔一秒钟刷新一次。在刷新 10 次后，浏览器中的信息就不再变化了。读者可以尝试着向数组中添加更多的内容，同时修改循环语句中的变量$i 的最大值，从而展现更加丰富的信息。

在这段脚本中，有很多的概念，比如数组、循环和条件判断等，都是大家第一次遇到。如果不懂，也没有关系，后续的章节中对这些内容进行详细地讲解。现在大家只用看看热闹就好。

3.4　习　　题

（1）请使用 echo 命令在浏览器中输入下面这一段文字：

```
"PHP 是一个免费开源的项目，无论是在 Web 服务器上部署 PHP 引擎，还是使用 PHP 代码编写网站，你都不需要花费一分钱。天下到底还是有免费的午餐。"
```

（2）请在刚才编写的脚本中使用"//"添加一段注释，注释内容如下：

```
"这是一条注释。"
```

（3）请将如下的内容以注释的形式添加到脚本中：

```
项目名称：向世界说你好
负责人：张小二
开始时间：2013 年 11 月 24 日星期日
结束时间：2013 年 11 月 25 日星期一
```

第 2 篇　常量、变量与数据

第 4 章　双生姐妹花——常量与变量

看到这个标题，也许很多同学都会好奇：难道要学代数吗？怎么又是变量，又是常量的呢？其实所谓的编程说白了就是代数加上一些条件判断和复用，它的核心就是代数。而常量和变量就是代数这棵树上结出的两朵并蒂双生花。

那么在进入这一章的主要内容之前，先来复习一下代数里关于常量和变量的定义吧。维基百科把常量定义为不变的量，其值是恒定不变的。与之相对的一个概念就是变量，其值会随着条件的改变而发生变化。在数学中，比较著名的常量包括：圆周率（π）、欧拉数（e）和黄金分割率（φ）。

掌握了这些基础知识之后，我们就明确了本章的主题。基于数学中常量与变量的定义，大家大概了解在 PHP 中的常量和变量所指为何。所谓常量应该也是恒定不变的量，而所谓变量则是随条件变化而发生变化的量。基于这样的认识，随后的两节内容将深入地介绍在 PHP 语境下的常量与变量到底是什么。

4.1　什么是常量

在 PHP 中，常量通常是一个包含固定值的量，包含在常量中的值不会随脚本中其他因素的改变而改变。当读者设定了一个常量的值之后，在脚本的任何地方引用这个值，其结果都是一致的。例如，在脚本中定义了一个常量 Weather，并将其值设定为 Sunny。那么在脚本中的任何地方引用 Weather，系统都会告诉我们天气晴朗。其实，Weather 更多的时候是当做变量使用的，因为天气是会变化的。这也是为什么中文里有那么多描绘气候变幻的成语了。

4.1.1　如何定义常量

在 PHP 脚本中，可以根据需要自行定义常量。例 4.1 所示展示了如何创建一个常量。

【例 4.1】 定义常量。

```
define ("constantname", "constantvalue");
```

例 4.1 所示的意思是用 define 方法创建一个名为 constantname，值为 constantvalue 的常量。在这里需要说明的是，define 方法是 PHP 定义常量的方法，只有通过这个方法才能在 PHP 中定义常量。这个方法带有两个参数：一个为常量名，另一个为常量值。对于常量名，请务必使用一对双引号将其包裹起来。对于常量值，请遵循如下规则：

- 若常量值为字符串，务必使用一对双引号将其包裹起来。

- 若常量值为数字，则无需使用双引号。

下面，来看看如何使用 define 方法把在 4.1 节的前言中提到例子写成脚本，如例 4.2 所示。

【例 4.2】 定义常量 WEATHER。

```
define ("WEATHER", "Sunny");
```

在例 4.2 中，WEATHER 就是常量名，而 Sunny 是常量值。无论在何处引用该常量，结果都是一样的。

需要进一步说明的是，常量名最好能很好地说明该常量指代的意义。同时，还要注意，常量名的每一个字母最好都大写，这样可以很好的与变量区分开。当然，无论是给常量取一个无意义的名字，还是全部用小写字母，对于 PHP 引擎来说都是一样的。其实，这样做并不是给 PHP 引擎看的，而是给我们自己看的。因此，为了更好地阅读代码，请务必使用具有描述性的词语给常量取名，并大写每一个字母。

在例 4.1 中，我们说到，如果常量值为数字时，常量值前后无须使用双引号。现在我们来定义一个常量 INTEREST_RATE，其值为 0.652，如例 4.3 所示。

【例 4.3】 定义 一个数字类型的常量。

```
define ("INTEREST_RATE", 0.652);
```

说到这里，大家应该基本掌握了如何定义常量。那么，在定义常量时，除了使用具有描述性的词语给常量命名以外，还有其他需要注意的吗？答案当然是肯定的。在定义常量时，千万不要使用 PHP 的命令关键字来给常量命令。否则，PHP 引擎会因为无法分辨常量与命令关键字而报错。来看一个例子。

【例 4.4】 用命令关键字 echo 来定义一个常量。

```
1 <?php
2 define ("ECHO", "Hello World!");
3 echo ECHO;
4 ?>
```

将例 4.4 所示的脚本保存为 PHP 脚本文件并加以执行，会有什么结果呢？图 4-1 所示展示了运行该脚本后的结果：

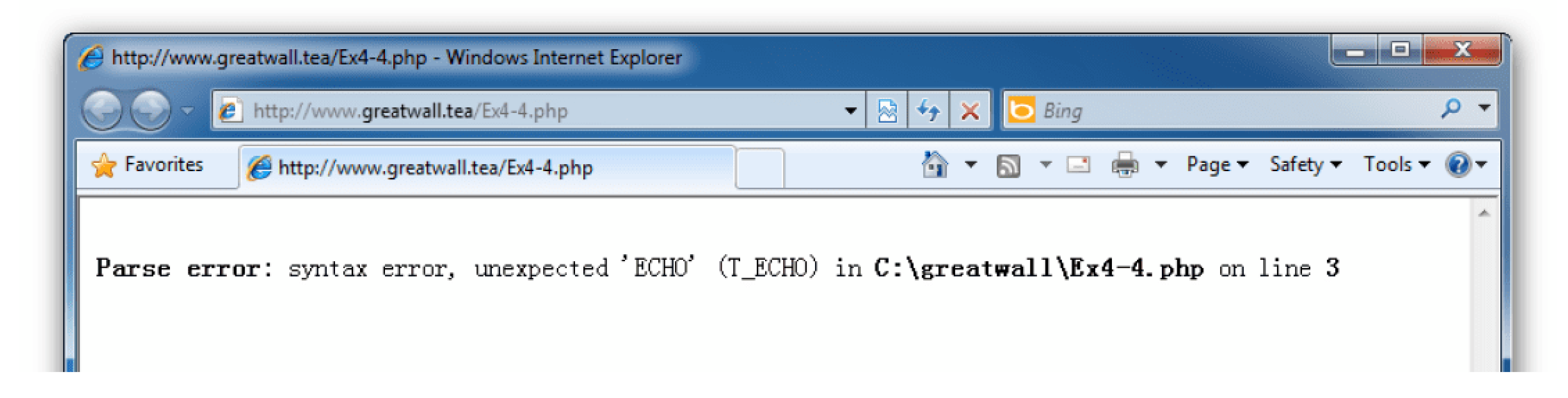


图 4-1　使用命令关键字定义常量的系统报错

根据系统提示，脚本出现了语法错误。在脚本的第三行出现了一个未知的 ECHO。这就表明了 PHP 引擎无法区分例 4.4 所示脚本中的两个 echo。

这样看来，在给常量取名时，务必要避开系统定义的命令关键字。表 4-1 所示列出了

一些常用的命令关键字：

表 4-1　常用的PHP命令关键字

序号	命令关键字	序号	命令关键字	序号	命令关键字
1	and	11	echo	21	list
2	as	12	else	22	new
3	break	13	empty	23	or
4	case	14	eval	24	print
5	class	15	exit	25	require
6	const	16	for	26	return
7	continue	17	foreach	27	switch
8	decline	18	global	28	use
9	die	19	if	29	var
10	do	20	include	30	while

4.1.2　何时使用常量

当某个值需要在脚本运行过程中恒定不变，就可以将其定义为一个常量。使用常量的好处是显而易见的。通过定义常量，可以用具有描述性的常量名来替代常量值，从而使得代码维护更加容易。例如，INTEREST_RATE 显然要比 0.652 好懂的多。

除此之外，当定义了一个常量之后，可以在脚本的任何地方引用该常量。当修改了该常量的值，所有引用该常量的地方也会同步更新修改后的常量值。因此，只需更新一处代码，即可实现脚本全局的更新。

下面还是用汇率来举例。假设美元对人民币的汇率为 6.22，我们可以写下如例 4.5 所示的脚本。

【例 4.5】 定义汇率为常量。

```
1    <?php
2    define ("INTEREST_RATE", 6.22);
3    $USD = 100;
4    $RMB = $USD * INTEREST_RATE;
5    echo $RMB;
6    ?>
```

在上面这个例子中，我们定义了一个常量 INTEREST_RATE，其值为 6.22。它代表了美元兑人民币的汇率。我们现在想知道手中的 100 美元相当于多少人民币。其值应该是拥有的美元数量乘以当前利率，为 622。为了计算这个结果，我们定义了两个变量：一个为美元（$USD），另一个为人民币（$RMB）。需要注意的是，在定义变量时，变量名前要加上变量定义符（$）。

然后，我们为变量$USD 赋值 100，表示我们拥有的 100 美元。按照之前的思路，变量$RMB 的值应该为$USD 乘以 INTEREST_RATE。于是，我们为$RMB 赋值$USD * INTEREST_RATE。

最后打印变量$RMB 的值。

例 4.5 中的这段脚本只有六行，看上去定义常量的作用不大。因为脚本只在第四行引用了该常量。但是通常脚本不会只有六行，常量的引用也不会只有一处。假设脚本有上千

行，对该常量的引用有数十处，定义常量的意义就显而易见了。

有的同学可能会问，可不可以把利率定义为一个变量呢，比如写成如例 4.6 所示的脚本。

【例 4.6】 定义汇率为变量。

```
<?php
$rate = 6.52;
$USD = 100;
$RMB = $USD * $rate;
echo &RMB;
?>
```

例 4.5 与例 4.6 的区别就在于前者定义了一个常量 INTEREST_RATE，并在计算人民币价值时引用了该常量；后者则定义了一个变量$rate，然后在计算人民币价值时引用了该变量。两者的其他代码均相同，得出的结果也是相同的。但是刚才也提到了，通常编写的代码肯定不止六行，假设在茫茫数千行代码中的某处，我们写下了如例 4.7 所示的代码。

【例 4.7】 重复定义变量$rate。

```
    ⋮
11  $rate = 80.31
    …
35  $RMB = $USD * $rate;
36  echo &RMB;
    ⋮
```

在例 4.7 中，我们在第 11 行重复定义了$rate，并为其赋了一个新值 80.31。之后，在第 35 行又一次引用了变量$rate。这时，计算出的$RMB 的值就变成了 8031，而不是之前的 622 了。而且在这行代码之后所有引用$rate 变量的地方，都会更新成 80.31，并被代入计算。

即便十分小心，也难保不会出现这样的错误。所以，对于一个短期内恒定不变或波动很小的值，将其定义为常量，是一个不错的选择。

4.1.3　PHP 预置常量

PHP 也为我们预定义了一些常量。比如常量__LINE__表示该常量所在的行号，常量__FILE__表示该常量所在的文件名。这两个常量生得有些奇怪：以两个下划线开始，再以两个下划线结束。大家可以通过如例 4.8 所示的脚本来检验这两个常量。

【例 4.8】 常量__LINE__和__FILE__。

```
<?php
    echo "Constant   LINE   has been used in line ",  LINE  ,".<br>";
    echo "The current file locates in ",  FILE  ,".";
?>
```

在例 4.8 所示的脚本使用 echo 命令输出了两句话。第一句话说的是常量__LINE__出现在脚本的第几行，第二句话说的是当前文件存放在什么路径下。我们运行该脚本可以得出如图 4-2 所示的结果。

根据截图中的内容，大家可以知道常量__LINE__所在的行和常量__FILE__所在文件的

存放路径。

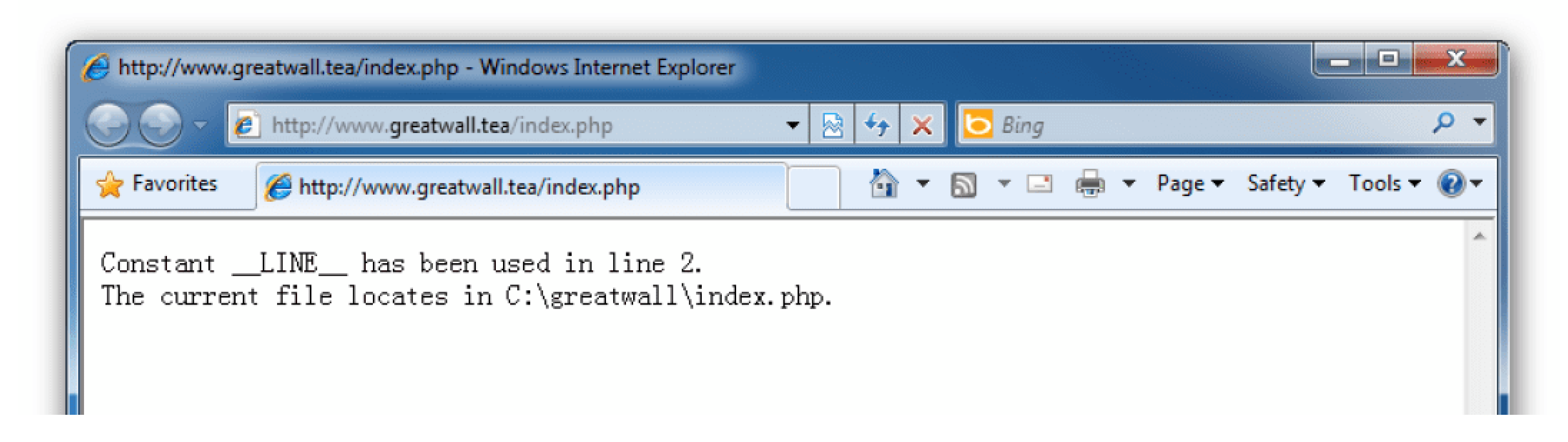


图 4-2　常量__LINE__和__FILE__

除了上述两个预置常量之外，PHP 还提供了一些用于处理脚本错误的常量，如 E_ALL 和 E_ERROR 等。这些常量可以影响 PHP 处理引擎如何处理脚本中出现的错误。

4.2　什么是变量

在上一节例 4.5 中，我们曾提到定义变量需要使用变量定义符（$）。这也说明了变量的定义方法与常量有着较大的区别。按照 PHP 脚本语言的语法规则，常量前不加任何符号，而变量前则必须带变量定义符。在本节里，我们就深入讨论一下变量。

4.2.1　变量的命名

现在大多数网站会要求大家注册，在注册时，需要填写诸多的信息。注册完毕后，大家使用注册的用户名和密码登录网站，会发现网站某个部分显示了我们注册的用户名。这一点相信大家并不陌生。

其实用户注册信息是通过变量在数据库和客户端之间传递的。在 PHP 脚本语言里，变量通常用来保存用户填写在表单里诸如姓名、年龄和性别之类的信息。之后，可以通过引用相应的变量来定制页面的内容。这样一来，每个用户看到的都是打上了不同个人标记的页面。

在了解如何定义变量之前，先来看看变量的命名规则。

和常量类似，变量也有变量名和变量值。不过在定义变量的时候，不需要使用 define 方法。与定义常量比较起来，定义变量有如下几点要求：

- 所有的变量名都必须以变量定义符（$）开头。变量定义符告诉 PHP 处理引擎接下来要处理一个变量。
- 变量名的长度没有限制，不过建议大家在为变量命名时使用具有描述性的词语。这一点要求与常量相同。
- 变量名只能使用字母、数字和下划线。其他类型的字符一律不能使用。
- 变量定义符后的第一个字符必须为字母或下划线，不能使用数字。
- 使用拼写相同的单词的大写形式和小写形式定义变量时，得到的是两个不同的变

量。比如$Car_Model 和$car_model 是两个不同的变量。

表 4-2 所示列出了一些合法的与非法的变量名，大家可以参照上面的规则来分析一下合法的为什么合法，非法的又为什么非法。

表 4-2　变量名举例

合法变量名		非法变量名	
$_name	$first_name	$3name	$name?
$name3	$name_3	$first+name	$first-name
$warnMsg	$salute_to_user	$key note	$salute to user

重点关注一下表 4-2 中列出的非法变量名。这些变量名之所以是非法的，要么是因为在变量定义符后的第一个字符处使用了数字，要么在变量名中使用了非法字符。使用这些变量名，系统会报错。以$first+name 为例来看一下 PHP 引擎在碰到这个变量名时的反应，如图 4-3 所示。

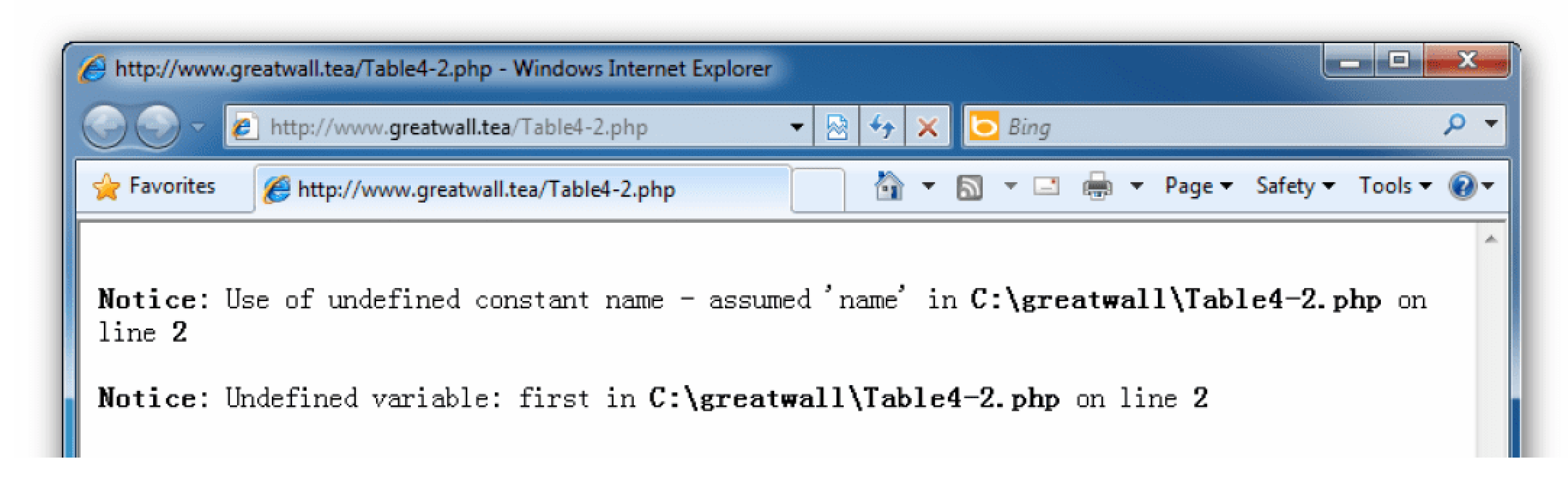


图 4-3　关于非法变量名$first+name 的报错信息

根据截图中的内容，可以发现 PHP 处理引擎将 name 当成了一个未定义的常量，而将该非法变量名中的“+”当成了运算符，进而认定$first 是一个未定义的变量。于是出现了图 4-3 中所示的两条报错信息。

另外，在表 4-2 所示中出现了如$warnMsg 和$first_name 这样的变量名，它们也是合法的变量名。由于在 PHP 脚本中，变量名的长度是没有限制的，因此可以用任何想用的字母组合来给变量命名。但是有一点，不能在变量名中使用除了下划线以外的其他符号。因此，如果想用两个或者更多单词来给变量命名时，要么在这些单词间使用下划线，要么把从第二个单词开始的每个单词的首字母大写。在本书中变量名的命名采用第二种方式。

这样做的理由同样也是为了方便今后阅读和维护脚本。读者也可以不这样做，不过在维护脚本时可能要费一些功夫来辨别每个变量代表的意思。

4.2.2　如何定义变量

在了解了变量的命名规则之后，再来看看如何定义一个变量。在讲变量的命名规则时，我们提到过定义变量是不需要使用 define 方法的。那么在 PHP 中应该如何定义一个变量呢。有一句话是这么说的：为变量赋值即定义一个变量。也就是说如果已经给一个变量分配了对应的变量值，那么这个变量是否存在就不再是个问题。

在 PHP 中，赋值用的是等号（=），叫做赋值符，意思是把赋值符右边的值分配给赋值符左边的变量。赋值符左右的内容不能互换，这与数学中的等号概念不同。例 4.9 所示展示了如何为变量赋值。

【例 4.9】 为变量赋值。

```
$name = "David Clark";
$age = 23;
$profession = "Assistance";
$salary = 8904.34;
```

在例 4.9 中定义了四个变量，有的变量值前后带有一对引号，有的则没有。带有引号的是字符串类型的变量。这一对引号告诉 PHP 处理引擎将引号内的全部内容当成一个值赋值给赋值符左边的变量。不带引号的是数字类型的变量。PHP 处理引擎在碰到这种类型的变量值时，会直接将其赋值给赋值符左边的变量。值得注意的是，无论是字符串型的变量，还是数字类型的变量，在 PHP 脚本中，都是可以计算的。

变量除了有不同的类型，还可以被重复定义。这一点在定义常量时是无法实现的。比如，可以运行一下例 4.10 所示的脚本。

【例 4.10】 重复定义一个常量。

```
<?php
    define ("AGE", 19);
    define ("AGE", 20);
?>
```

运行后，系统会报错。报错信息如图 4-4 所示，常量 AGE 已经定义了，不能再次定义。如果需要修改常量的值，则务必在定义该常量的语句中修改。

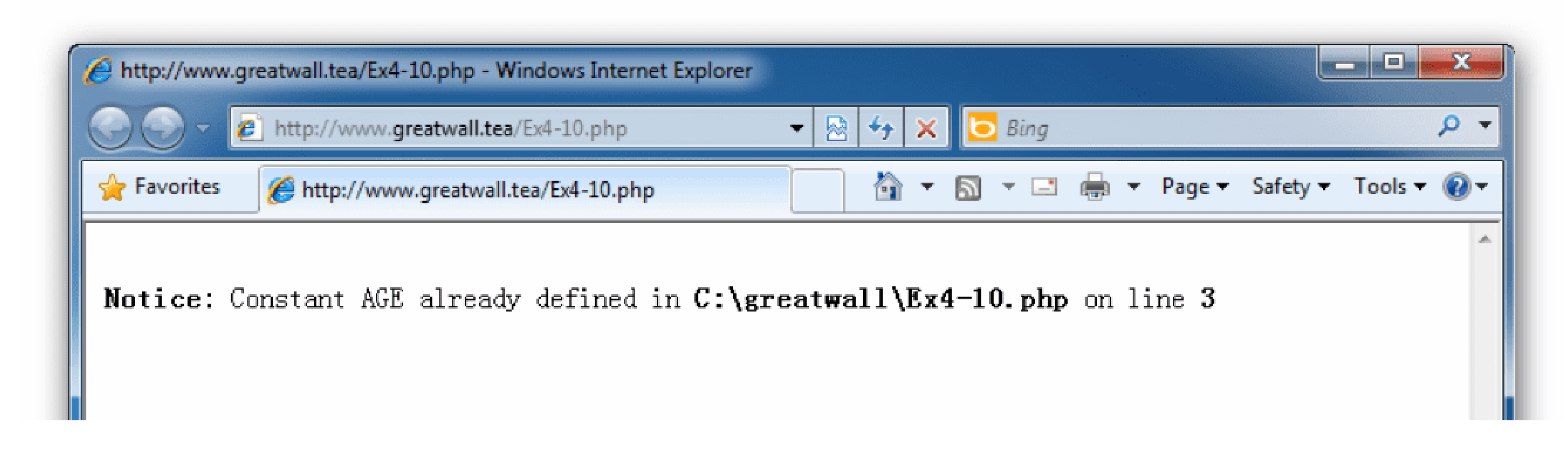


图 4-4　重复定义常量时的报错信息

对于变量而言，重复定义是十分普遍的。再来看一段脚本，如例 4.11 所示。

【例 4.11】 重复定义一个变量。

```
<?php
    $color = "blue";
    $color = "red";
    echo "The color is $color";
?>
```

这段脚本的运行结果输出了第二次我们为变量$color 赋的值。这也说明，变量是可以被重复定义的。需要注意的是，一旦再次定义了某个变量，该变量的值就发生了变化，之前为其赋的值就被最新赋的值覆盖了。

值得注意的是，在这段脚本的第四行，我们把变量名$color 放在了引号里面，这样它就可以和引号中的其他信息一起输出。但是把变量名写在引号里面有时也会存在问题。下面来看一个例子。

【例 4.12】 在引号中使用变量（一）。

```
<?php
    $bodyPart = "tooth";
    echo "You have a $bodyPartache.";
?>
```

其实脚本的初衷是想让 PHP 处理引擎输出“You have a toothache.”这句话，结果它给了我们一记白眼，输出了下面这句话：

```
Notice: Undefined variable: bodyPartache in C:\greatwall\Ex4-12.php on line 3
```

原来 PHP 处理引擎把 bodyPartache 当成了一个变量，而脚本中又没有定义这个叫 bodyPartache 的变量，所以出现了错误。那么有没有办法可以修改一下呢？答案是肯定的。其实，在引号中使用的变量前后加上一对花括号（{}）就可以实现目的。因此，可以把例 4.12 修改成如下的样子：

【例 4.13】 在引号中使用变量（二）。

```
<?php
    $bodyPart = "tooth";
    echo "You have a {$bodyPart}ache.";
?>
```

还可以把一个变量做为值赋给另一个变量。这一点也是常量无法做到的，如例 4.14 所示。

【例 4.14】 把变量做值赋给另一个变量。

```
<?php
    $user = "Sally";
    $salute = "Good Morning! ";
    $saluteToUser = $salute.$user;
    echo $saluteToUser;
?>
```

这段脚本定义了三个变量，分别为$user、$salute 和$salueToUser。其中$saluteToUser 的值为变量$salute 和$user 的组合。输出的结果就是这两个变量的值的组合。

4.2.3　详谈变量输出

到这里，我们已经使用 echo 命令编写了几个脚本输出了一些变量的值。可能对于某些同学来说，通过这些脚本掌握的知识是零散的。为了巩固对输出变量的认识，下面把可能会见到的变量输出方式做个总结。

表 4-3 所示中使用的变量定义如下：

```
$number = 159;
$word1 = "Hello";
$word2 = "World";
```

表 4-3　使用echo命令输出变量及其结果示例

Echo 命令	输 出 结 果
echo $number;	159
echo $word1,$word2;	HelloWorld
echo $word1," ",$word2;	Hello World
echo 'Say "$word1 $word2" now!';	Say "Hello World" now!

说到变量的输出，再啰嗦两句。其实 PHP 提供了不止 echo 一种输出信息的方法。还有若干种其他的输出信息的方法。其中一种叫做 print_r 方法，它的功能是输出关于变量易于人理解的信息。对于单个变量来说，无论该变量是字符串、整型还是浮点数变量，使用 print_r 方法都可以方便地输入该变量的值，这一点与 echo 命令一样。但是与 echo 命令不同的是，print_r 一次只能输出一个变量的值，而 echo 一次可以输出多个变量的值。为了说清楚这个问题，来看一段脚本。

【例 4.15】 print_r 和 echo 的区别。

```
<?php
    $user = "Sally";
    $salute = "Good Morning! ";
    print_r($user);
    print_r($salute);
    echo $salute,$user;
?>
```

例 4.15 使用 print_r 方法输出了$user 和$salute 两个字符串变量的值，分别为“Sally”和“Good Morning!”。然后使用 echo 方法输出了这两个变量值组合在一起的结果。

使用 echo 命令输出变量值的时候，如果引用未定义的变量，系统会报错。比如脚本中定义了一个变量$age，结果在 echo 命令中引用时写成了$aeg。这时系统会出现如下所示的错误。

```
Notice: Undefined variable: aeg in C:\testvar.php on line 5
```

读者可以通过在变量引用的变量名前加@来关闭系统报错（如 echo @$aeg）。这样系统就不会显示上述出错信息。但是由于$aeg 的确是一个未定义的变量，所以系统也不会输出任何信息，你也不会看到任何结果。

此处的建议是，如无必要，千万不要关闭任何出错信息。不是有句话这么说的么：出了错不可怕，可怕的是出了错自己都不知道错在哪儿。

知错能改，善莫大焉。

4.2.4　何时使用变量

在知道如何定义变量和输出变量之后，还需要知道何时使用变量。与常量的使用类似，当某种信息会由于条件的不同而发生变化，那么使用变量是最合适不过的了。例如，“天气”通常被看成一个复杂多变的概念，人们的活动会随着天气的不同而发生变化。那么，可以编写一段 PHP 脚本来描述这种变化。

【例 4.16】 示例脚本：人与天气。

```
<?php
```

```
    //定义变量$weather
    $weather = "sunny";

    //描述在不同天气条件下人的活动方式
    switch ($weather) {
        case "sunny" :
            echo "It's ",$weather,", you can take a walk in the sunset.";
            break;
        case "rainy" :
            echo "It's ",$weather,", you should take an umbrella with you.";
            break;
        case "windy" :
            echo "It's ",$weather,", you should wear your jacket.";
            break;
    }
?>
```

例 4.16 定义了一个变量$weather，并且描述了在不同天气情况下，人们一般会从事的活动。通过修改变量$weather 的值，脚本会输出不同的描述片段。这里可以使用 switch 方法。它和第 2 章里见到的 if 命令类似，都是用于描述条件的。翻译成自然语言就是：如果出现了 A 状况，如何如何；如果出现了 B 状况，如何如何。关于 switch 方法，我们会在第 8 章中详细讲到。

在 PHP 脚本中，使用变量的情况要比使用常量的情况多的多。可以说大多数情况下，都需要使用变量。这不光是因为变量定义起来很方便，还因为与常量比较起来，变量更加灵活。再来看一个例子。

【例 4.17】 用变量 A 的值充当变量 B 的名：显示相关城市人口数量（单位：万人）。

```
<?php
    //使用城市名称定义若干变量，用于存储各城市的人口数量
    $shanghai = 2231.5;
    $beijing = 1882.7;
    $chongqin = 1569.4;
    $tianjin = 1109.0;
    $guangzhou = 1107.1;

    //定义一个变量用于存储城市名称
    $cityName = "beijing";

    //输出北京市的人口数量
    echo "The population in $cityName is ${$cityName}.<br>";

    //修改$cityName 的值为广州市
    $cityName = "guangzhou";

    //输出广州市的人口数量
    echo "The population in $cityName is ${$cityName}.";
?>
```

例 4.17 输出的结果如图 4-5 所示。

例 4.17 分别用五个城市的名字定义了五个变量，用来存储这五个城市的人口数量。然后又定义了一个变量用来指定这五个城市中的某一个城市，最后使用 echo 命令输出关于该城市的人口数量的一句描述。

大家可以看到例子中有两条 echo 命令。它们都是一样的。

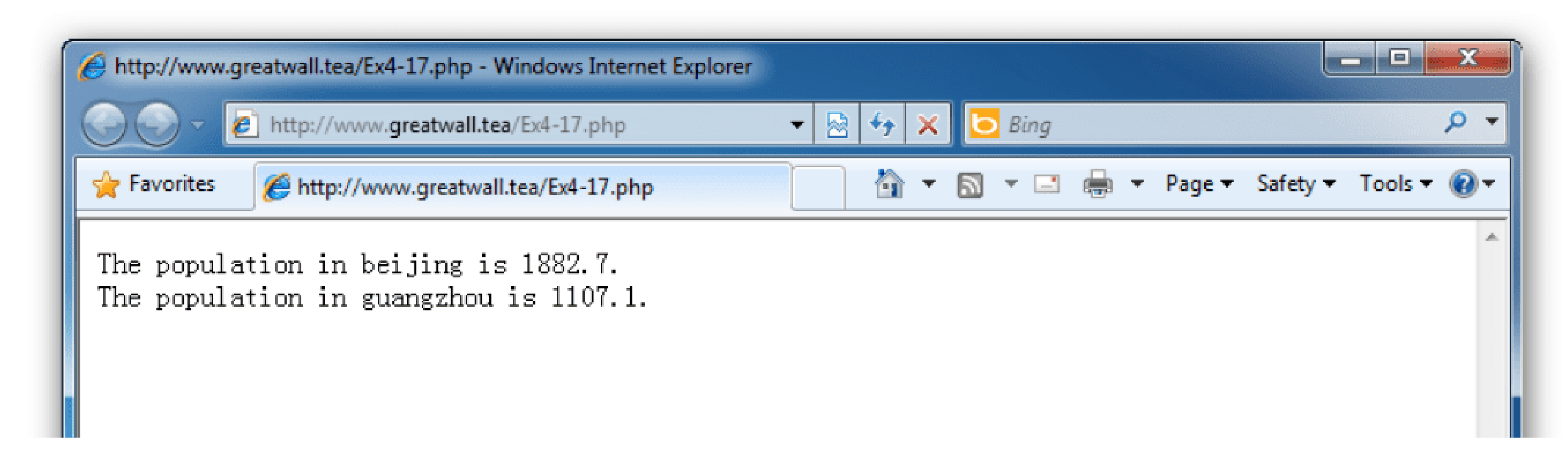


图 4-5　用变量 A 的值充当变量 B 的名：输出结果

```
echo "The population in $cityName is ${$cityName}.<br>";
```

在这条 echo 命令中，变量$cityName 使用了两次。但是根据输出的结果来看，第一次输出了城市的名字，而第二次却输出了城市的人口数量。这是为什么呢？原来在第二次引用变量$cityName 时，在它的前面又加了一个变量定义符（$）。当指定变量$cityName 的值为 beijing 时，$$cityName 就等价于$beijing。当指定变量$cityName 的值为 guangzhou 时，$$cityName 就等价于$guangzhou。因为 PHP 处理引擎在碰到$$cityName 时，会认为两个变量定义符（$）后面跟着的都是变量名，于是把$cityName 转换成了它对应的值，然后又把以$cityName 对应的值为变量名的变量转换成了它对应的值。

在这个例子中，正是有了这样的两次转换，才在两次引用同一变量的情况下，输出了不同的值。

4.2.5　如何销毁变量

在定义了某个变量之后，如果一旦不需要这个变量了，可以销毁它。来看下面这个例子。

【例 4.18】 销毁变量。

```
<?php
    //定义一个变量$age
    $age = 10;

    //输出变量$age
    echo $age;

    //销毁变量$age
    unset ($age);

    //再次输出变量$age
    echo $age;
?>
```

大家猜测一下，在运行这段脚本后，浏览器会有什么反应呢？图 4-6 所示展示了浏览器的反应。

根据图 4-6 显示的结果，当脚本第一次要求输出变量$age 的时候，PHP 处理引擎照做了。但是当使用 unset 方法销毁变量$age 后再次输出时，PHP 处理引擎又给了我们一记白眼，输出了错误提示：

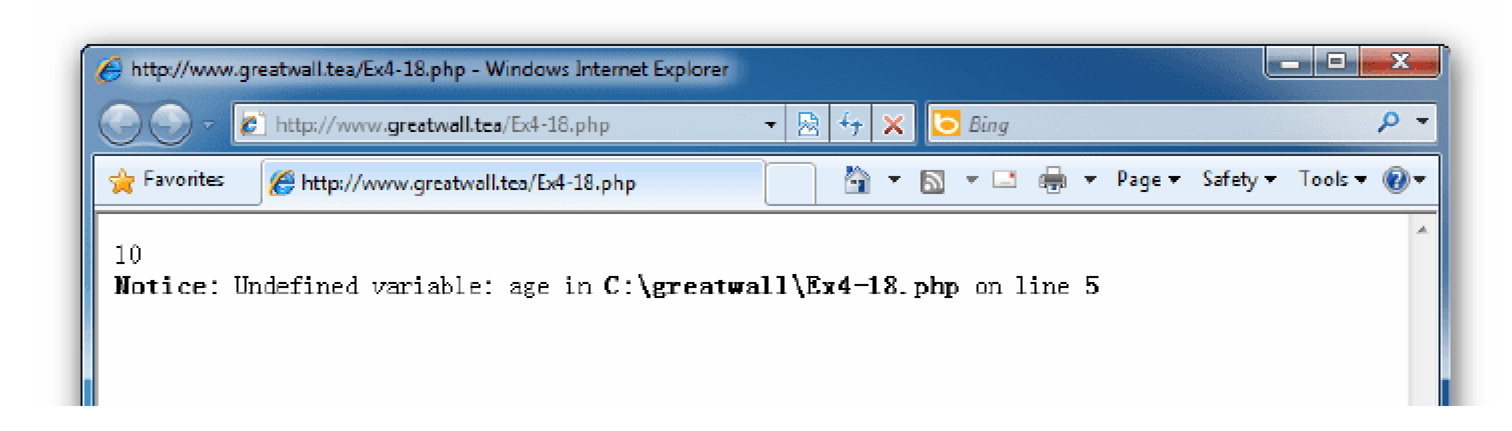


图 4-6　销毁变量

```
Notice: Undefined variable: age in C:\greatwall\Ex4-18.php on line 5.
```

这条错误提示说明变量$age 此时已经不复存在了。

4.3　实战练习：常量与变量

4.3.1　背景介绍

本节将使用本章中学到的知识来编写一个货币兑换的脚本。在这里，以人民币为基础货币，同时以 2013 年 7 月 5 日单位人民币兑换各主要货币的汇率为常量，开展货币兑换的业务，如图 4-7 所示。

今日外汇牌价最新查询（2013年7月5日）					
币种	交易单位	中间价	现汇买入价	现钞买入价	卖出价
美元 (USD)	100	612.50	611.28	606.38	613.73
港币 (HKD)	100	78.99	78.83	78.28	79.15
欧元 (EUR)	100	789.94	786.78	761.90	793.10
英镑 (GBP)	100	921.45	917.76	888.74	925.14
日元 (JPY)	100	6.10	6.08	5.89	6.13

图 4-7　2013 年 7 月 5 日外汇牌价

货币兑换平台看上去如图 4-8 所示的样子。为此，需要先用 HTML 对货币兑换平台进行装修。

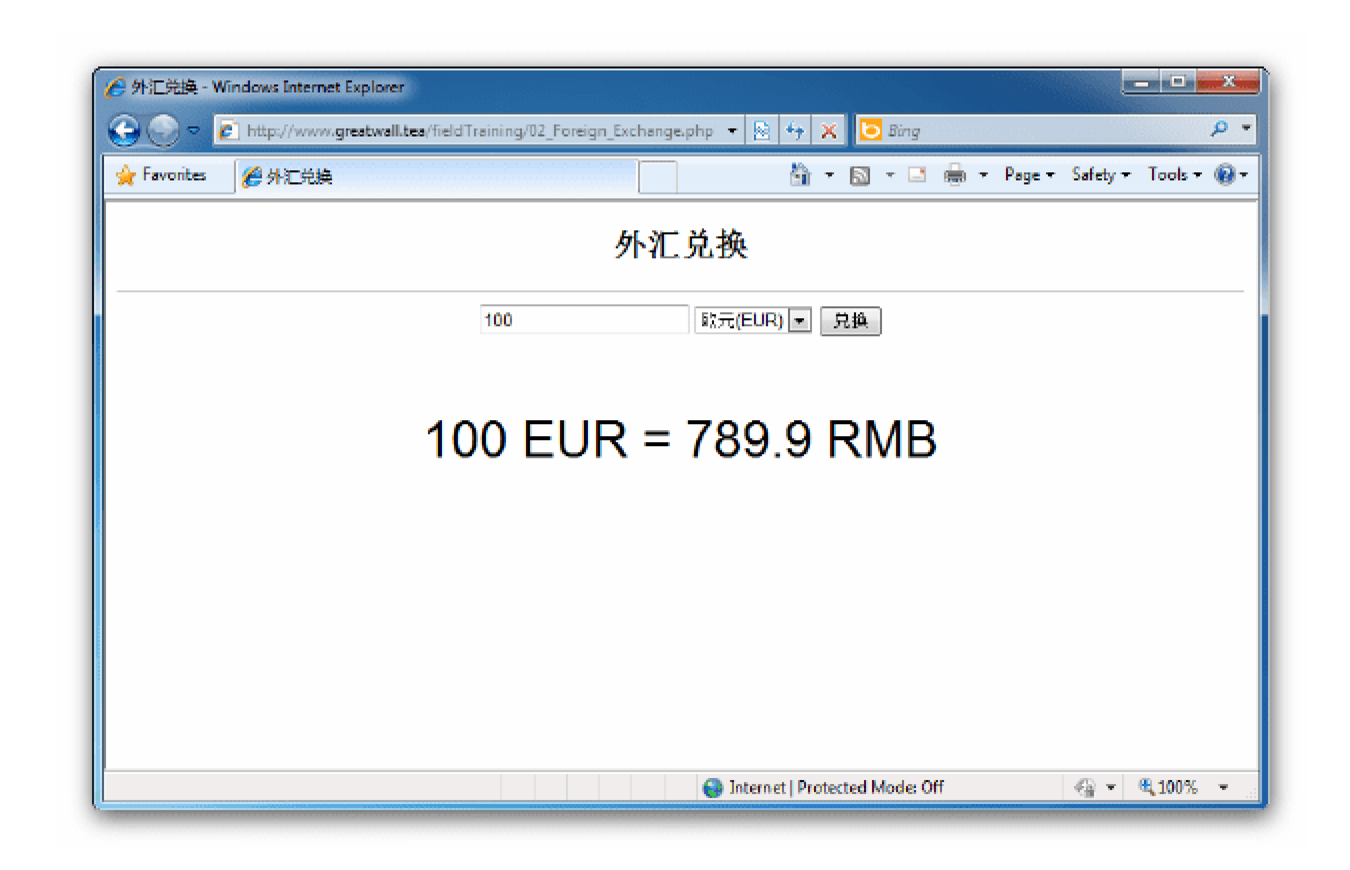


图 4-8　货币兑换平台

4.3.2　实现过程

HTML 代码如下：

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>外汇兑换</title>
        <style type="text/css">
            #msg {
                line-height:150px;
                text-align:center;
                font-size:230%;
                font-family:sans-serif;
            }

            form, h2 {
                text-align:center;
            }
        </style>
    </head>
    <body>
        <h2>外汇兑换</h2>                                //货币兑换处的标题
        <hr />
        <form action='?check' method="post">
            <input type='text' name="quantity">   //用户输入指定外汇数量的文本框
            <select name="currency">        //用户选择指定外汇各类的下拉框
                <option value='USD'>美元(USD)</option>
                <option value='HKD'>港元(HKD)</option>
                <option value='EUR'>欧元(EUR)</option>
                <option value='GBP'>英镑(GBP)</option>
                <option value='JPY'>日元(JPY)</option>
            </select>
            <button type="submit">兑换</button>     //用户提交表单的按钮
        </form>
        <div id="msg"></div>                            //显示兑换结果的 div
    </body>
</html>
```

这段 HTML 代码使用了一张表单，并定义了一个显示兑换结果的 DIV。然后使用了一段简单的 CSS 代码对页面进行了装修。值得注意的是，为了在当前页面处理用户通过表单提交的信息，表单的 action 属性的值设置成了“=?check”。然后通过 PHP 脚本根据 action 属性的值来判断用户是否提交了表单。

关于页面间值的传递会在第 9 章中讲到，这里就先忽略过去了。

接下来，需要在“</body>”标签前面添加一段 PHP 脚本，来使表单交互起作用。注意，添加 PHP 脚本的地方一定要是“</body>”标签前，否则是看不到脚本执行的结果的。

需要添加的 PHP 脚本如下：

```
<?php
    if (isset($_REQUEST['check'])){         //判断用户是否已经提交了
        define('USD', 6.125);
        define('HKD', 0.789);
```

```
        define('EUR', 7.899);
        define('GBP', 9.214);
        define('JPY', 0.061);   //定义五个常量分别对应五种主要货币兑人民币的汇率

        $quantity = intval($_REQUEST['quantity']);//获取用户需兑换的外汇数量
        $currency = $_REQUEST['currency'];      //获取用户所持有的外汇种类

        switch($currency)           //根据获取的外汇种类，计算等值人民币数量
        {
            case 'USD':
                $amount = $quantity * USD;
                break;
            case 'HKD':
                $amount = $quantity * HKD;
                break;
            case 'EUR':
                $amount = $quantity * EUR;
                break;
            case 'GBP':
                $amount = $quantity * GBP;
                break;
            case 'JPY':
                $amount = $quantity * JPY;
                break;
        }

        echo '<script ge="javascript">document.getElementById("msg")
        .innerHTML="'.
        $quantity.' '.$currency.' = '.$amount.' RMB'.
        '"</script>';              //输出最后结果到 ID 为 msg 的 div 中
    }
?>
```

这段脚本通过$_REQUEST 这样一个系统预置的常量数组来判断当前页面是否定义了一个索引为“check”的数组元素。值的注意的是，在判断的时候使用了 isset()函数，即判断系统中是否设置了这个“$_REQUEST['check']”数组元素。若设置了，则表明用户提交了表单；若没有设置，则表明用户没有提交表单。

这里提到的$_REQUEST 常量数组其实就是在页面间传递数组时经常用到的一个数组。与它类似的两个常量数组分别是$_POST 和$GET。关于这三个系统预置的常量数组，大家可以参见第 9 章的内容。

系统在检查到用户提交了表单之后便开始执行 PHP 脚本 if 语句中的内容。在 if 语句中，脚本分成了四个步骤。它们分别是：

（1）定义五个常量，分别用来存储五种货币与人民币的汇率。

（2）获取用户在表单中填写的需要兑换的货币数量和货币类型。

（3）根据获取到的货币数量和货币类型计算可兑换的人民币数量。

（4）将计算结果输出到浏览器指定位置。

第一步使用 define()函数定义了五个常量用来存储五种货币与人民币的汇率。需要记住的是，通过 define()函数定义的常量的值在整个脚本中都不会发生改变，该常量也不能再次被定义。

第二步继续使用$_REQUEST 这个常量数组获取用户通过表单提交的内容，其中$_REQUEST['quantity']这个数组元素存储着用户在表单中填写的外币数量，而$_REQUEST

['currency']这个数组元素存储着的则是用户在表单中选择的货币类型。细心的读者可能已经发现了，这两个数组元素的索引正是表单中定义的文本框元素和下拉框元素的“name”属性的值。由于获取的$_REQUEST['quantity']这个元素的值为字符串类型的值，因此，脚本使用了 int()函数将其转换成一个整型的值，以便后续的计算。

第三步根据获取的外汇类型和数量，计算可兑换的人民币数量。在这里需要注意的是，脚本使用了 switch 语句根据货币类型来控制脚本执行的顺序。即当货币类型为“HKD”，执行“case 'HKD':”部分的脚本，并在见到“break;”语句时，退出 switch 语句。通过这种方式，就可以获取基于用户输入的信息计算的可兑换的人民币数量。

第四步使用 echo 语句输出一段 JavaScript 代码。这段代码将之前的计算结果按照定义的格式输出到浏览器的指定位置上。

当在浏览器中打开这个页面，并在文本框中输入一个任意的正数，系统就会在指定的位置返回指定数量的外汇可兑换的人民币数量。

本例使用了 switch 语句控制脚本的执行顺序、演示了如何定义常量和变量，以及如何使用它们进行运算来得出想要的结果。关于 switch 语句的介绍，大家可以参见第 7 章的内容。

4.4　习　　题

（1）根据脚本中的注释完成任务。

```
<?php
    //请在下方添加脚本中缺失的部分

    $total = RATE * $amount;
    echo $total;
?>
```

（2）根据脚本中的注释完成任务。

```
<?php
    $a = 1;
    $b = 2;
    $c = $a + b;
    //请在下方添加一段脚本，使 echo 语句输出一段错误提示

    echo $c;
?>
```

（3）请编写一段脚本，输出脚本文件在服务器上的路径。

第 5 章　数据五虎将

在上一章里，我们提到了数据类型这个概念。数据类型，顾名思义，应该就是数据是什么类型。常量和变量都可以用来存储不同类型的数据，不同类型的数据可以做不同的事情。例如，可以把都是数字类型的变量加起来求和，但是把都是字符串类型的变量相加却没有什么意义。在本章里，我们要讲一讲 PHP 支持的数据类型以及它们的用法。

5.1　概　　述

在这一节里，主要介绍各种数据类型的特点，并掌握如何为变量指定数据类型。

5.1.1　数据全家福

看到本章的标题，各位同学应该也能推测出来：PHP 支持的数据类型应该有许多种。现在我们按照它们的特点和出场频率逐一介绍一下：

- 整型数据是五兄弟中的排头老大，在数学方面天资不错，加减乘除全会算。它的能力根据安装有 PHP 引擎的服务器使用的操作系统的不同会发生变化。一般来说，正负二十亿范围内的四则运算交给它肯定没有问题。
- 浮点型数据在五兄弟中排行老二，在数学方面有着惊人的天赋，它会算各种复杂的小数运算。正弦、余弦、正切、余切、平方根和微积分什么的交给它一点问题也没有。虽然它的能力也会根据服务器使用的操作系统的不同而发生变化，不过通常情况下，输出指定位数的小数值，那你就擎好吧。
- 字符串型数据在五兄弟中排行老三，平时好舞个文、弄个墨。诗词歌赋什么的，只要教给它的，它能正着背过来，倒着背过去，还不带一个错字儿的。真真儿的一个学富五车的翩翩俏公子。
- 布尔型数据在五兄弟中排行老四，是个正儿八经的法官。它最常念叨的一句台词就是“to be or not to be，that is a question”。别看它平时和哈姆雷特一样神叨叨的，让它给断个案子什么的，它一看一个准儿。它的本事可大了，只要搂上一眼，能立马告诉你什么是真、什么是假、什么是对、什么是错。
- 时间日期型数据在五兄弟中排行老幺，也没其他别的爱好，就好摆个钟、弄个表。时间是怎么从指缝中溜走的，它了然于胸。千万不要没事儿就问他几点了，它的答案会把你烦死。你问为什么？因为它告诉你的时间有零有整，一说一长串，半天也停不下来。

有了这五位兄弟做你的左膀右臂，你的 PHP 脚本肯定会如虎添翼，变得多姿多彩起来。

知道它们各自的能耐之后，我们来看看，如何把它们运用到脚本里。

5.1.2　为变量指定数据类型

PHP 是一种弱类型的脚本。也就是说，你不需要刻意地为变量指定数据类型。就像在上一章中所做的那样，为一个变量赋了一个什么类型的值，这个变量就是什么类型的变量。因为 PHP 处理引擎可以根据变量的值自动判断该变量的数据类型，同时还会根据实际情况来变更数据的类型。来看一段脚本。

【例 5.1】 为变量指定数据类型。

```
$firstNumber = 1;                    //PHP 处理引擎自动认定该变量为整型变量
$secondNumber = 1.1;                 //PHP 处理引擎自动认定该变量为浮点型变量
$thirdNumber = $firstNumber + $secondNumber;
```

在例 5.1 所示的脚本里，定义了两个变量$firstNumber 和$secondNumber，并为$firstNumber 赋了一个整型值 1，为$secondNumber 赋了一个浮点值 1.1。然后定义了第三个变量$thirdNumber，用来存储前两个变量的和。要是放在其他的编程语言里，直接把这两个不同类型的变量直接相加是不可能的。但是在 PHP 里就不同了，PHP 处理引擎可以自动将整型变量转换成浮点型变量，然后再把两个变量相加以后的值赋给第三个变量。这样一来，第三个变量也就成了浮点型的变量了。

但是话又说回来了，PHP 处理引擎也不是回回都能认准变量的数据类型。在这种情况下，就得告诉 PHP 引擎想存储什么类型的数据，而不是让 PHP 处理引擎自己去猜测。在脚本中可以用如下的语句来为变量指定数据类型。

【例 5.2】 为变量指定数据类型。

```
$newInt = (int) $var1;               //指定变量$var1 的数据类型为整型
$newFloat = (float) $var1;           //指定变量$var1 的数据类型为浮点型
$newString = (string) $var1;         //指定变量$var1 的数据类型为字符串型
```

通过使用类似例 5.2 中的语句，就可以将等号右边的值按照括号中指定的数据类型存储到等号左边的变量中。可是，即使是使用指定数据类型也不见得就万无一失。要是当了乔太守，乱点了鸳鸯谱，出了错连找都找不着哪儿错了。因为 PHP 处理引擎不会对你的错误报错的。比如下面的示例。

【例 5.3】 为变量指定错误的数据类型。

```
$var1 = 1.8;
$var2 = 2;
$newVar1 = (int) $var1;
$sum1 = $var1 + $var2;
$sum2 = $newVar1 + $var2;
```

在如例 5.3 所示的脚本中，读者觉得变量$sum1 和$sum2 的值会是多少呢？是 2.8 呢，3 呢，还是 4 呢？

先来分析一下这五个变量的数据类型：$var1 为浮点型变量，$var2 为整型变量，$newVar1 为整型变量。按照 PHP 处理引擎的习惯，$sum1 应该是个浮点型的变量，而$sum2 则是个整型变量。

再来分析一下变量$sum1 和$sum2 的值：按说$sum1 的值肯定是 2.8，这个没有问题。关键是如果$newVar1 的值如果还是 1.8 的话，那就不可能是个整型数据了，如果$newVar1 是个整型数据的话，那它的值就肯定不是 1.8。如果在第三行后输出$newVar1 的值，你会惊讶的发现，$newVar1 的值居然是 1。也就是说 PHP 处理引擎对变量$var1 做了取整处理。这样一来，$sum2 的值就成了 2。

想一想，如果在脚本中随意指定了变量的数据类型会发生什么问题吧。几千行的代码，系统也没有报错，那就只能一行一行的找了。所以说，不到万不得已，千万不要自行指定数据类型，一旦指定错误，恐怕你真得去面壁了。

5.2　玩转数字——整型和浮点型数据

按照之前的说法，可以通过为变量赋值来定义变量。给变量赋什么类型的值，变量就是什么类型的变量。无论是整型数据，还是浮点型数据，都是数字类型的数据。它们可以通过以下语句赋值，PHP 处理引擎会自动识别其数据类型。

【例 5.4】 定义整型和浮点型变量。

```
$intVar = 1;
$floatVar = 2.6;
```

5.2.1　四则运算

在 PHP 中，可以用数学运算符连接两个数字或者数字类型的变量来对数字类型的数据进行四则运算。比如，使用加号（+）连接两个数字，PHP 处理引擎会给出这两个数字相加的结果，如：

```
1 + 2
```

也可以用运算符连接两个变量，如：

```
$var1 + $var2;
```

需要说明的是，如果需要对变量的值进行四则运算，千万不要在定义变量的时候用引号把变量的值包裹起来。否则 PHP 处理引擎会将其自动识别为字符串型的数据，进而在进行某些四则运算的时候出错。来看下面一段脚本。

【例 5.5】 字符串与数字相加的结果（一）。

```
$intVarOne = "1";
$intVarTwo = 2;
$totalA = $intVarOne + $intVarTwo;
echo "The value of total A is ".$totalA.".<br>";

$varOne = "x";
$totalB = $varOne + $intVarTwo;
echo "The value of total B is ".$totalB.".";
```

那么 PHP 会怎么处理这段脚本呢？我们知道 PHP 处理引擎在运算时会根据变量的实际情况来自动转换变量的数据类型，那么在计算变量$totalA 的值时，PHP 是会将变量

$intVarOne 的值转换成一个整型数据呢，还是会将变量$intVarTwo 的值转换成一个字符串呢？如果是把变量$intVarOne 的值转换成一个整型数据的话，那在计算变量$totalB 的时候，变量$varOne 的值又会转换成什么呢？在回答这些问题之前，我们先来看看结果，如图 5-1 所示。

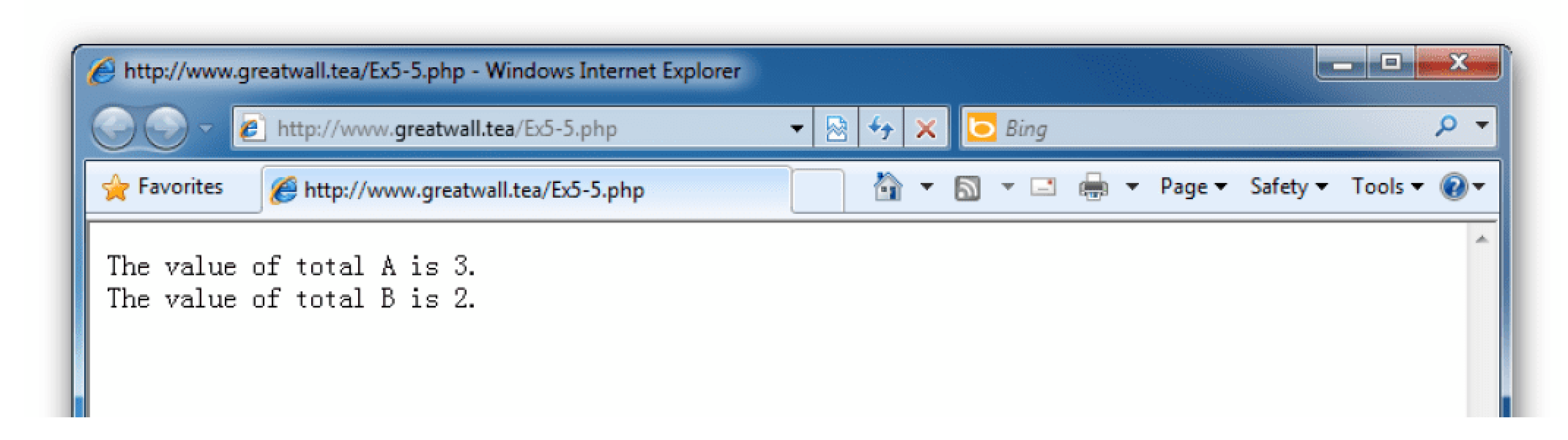


图 5-1　字符串和数字相加的结果

我们发现变量$totalA 的值为 3，也就是说 PHP 处理引擎在计算变量$totalA 的值时，自动将变量$intVarOne 的值由字符串型数据转换成了整型数据。但是有些看不懂的是，为什么变量$totalB 的值是 2 呢？难道说 PHP 处理引擎自动将变量$varOne 的值判定为 0 了不成？

的确是这样。因为变量$varOne 的值不是 PHP 处理引擎想转换就能转换得了的。对于这种无法转换但是又必须代入计算的时候，PHP 处理引擎就将其值转换为整型数据 0，然后再代入计算。

虽然说 PHP 处理引擎自动转换数据类型的功能很强大，但是千万不要认为它回回都能转换出你想要的数据类型。再来看一段脚本。

【例 5.6】 字符串与数字相加的结果（二）。

```
$intVarOne = "2,000";
$intVarTwo = 12;
$totalA = $intVarOne + $intVarTwo;
echo "The value of total A is ".$totalA.".<br>";
```

大家可以先猜测一下：这段脚本的运行结果到底是不是 2012 呢？

我既然这么不怀好意地问大家，答案自然是否定的。这段脚本的运行结果，也就是变量$totalA 的值，其实是 14。为什么呢？

因为 PHP 处理引擎不认识数字断位符，它在进行数据类型转换时看到逗号就停下来了，于是$intVarOne 的值被自动转换成了整型类型的 2。这样代入计算的话，变量 totalA 的值自然是 14，而不是 2012 了。

当然，PHP 支持的数学运算符不是只有加号（+），表 5-1 列出了 PHP 支持的五种数学运算符。

表 5-1　PHP支持的数学运算符

运算符	描　述
+	取两数或两变量之和，如“echo $var1 + $var2;”或“echo 1 + 2;”，其值为 3
-	取两数或两变量之差，如“echo $var1 - $var2;”或 “echo 3 - 1;”，其值为 2
*	取两数或两变量之积，如“echo $var1 * $var2;”或“echo 2 * 2;”，其值为 4

续表

运算符	描　　述
/	取两数或两变量之商，如“echo $var1/$var2;”或“echo 4/2;”，其值为 2
%	取两数或两变量之余，如“echo $var1%$var2;”或“echo 5%3;”，其值为 2

表 5-1 所示中列出的五种运算符中，第五种是用来求两数的余数。前四种就相当于数学里的加减乘除。在使用它们进行四则运算时也遵守加减乘除的法则，即按照先乘除后加减的顺序从左往右依次计算。在书写运算式时，也可以使用括号，括号里的内容优先计算。这些都是我们在学小学数学时都已经掌握了的。下面来看几个例子。

【例 5.7】 四则运算。

```
$var = (1 + 2) *3 + 1;        //第一步，先计算括号内的加法
$var = 3 * 3 + 1;             //第二步，计算乘法
$var = 9 + 1;                 //第三步，计算加法
$var = $var + 1;
```

对于像“$var = $var + 1”这样的句子，可以简写成$var++。类似的，像“$var = $var - 1”这样的句子，可以简写成$var --。

【例 5.8】 PHP 脚本中的缩略句（一）。

```
$var = 0;
$var++;
echo $var;       //此时输出的变量$var 的值为 1，相当于 echo $var + 1;
$var--;
echo $var        //此时输出的变量$var 的值为 0，相当于 echo $var - 1;
```

对于在某变量自身上或加、或减、或乘、或除一个常数的情况，PHP 脚本都提供了缩略句。比如例 5.9。

【例 5.9】 PHP 脚本中的缩略句（二）。

```
$var = 10;
$var +=2;
echo $var;       //此时输出的变量$var 的值为 12，相当于 echo $var +2;
$var -=3;
echo $var   ;    //此时输出的变量$var 的值为 9，相当于 echo $var - 3;
$var *=2;
echo $var;       //此时输出的变量$var 的值为 18，相当于 echo $var * 2;
$var /=3;
echo $var;       //此时输出的变量$var 的值为 6，相当于 echo $var / 3;
```

5.2.2　复杂运算

这里说的复杂运算其实也复杂不到哪儿去。PHP 提供了很多的函数用于诸如求平方根之类的运算。这样一来，就省去了写脚本的麻烦。只用写上一句简单的语句就能实现这类运算。关于函数的内容，我们会在第 9 章中讲到。现在来看一个例子。

【例 5.10】 求平方根。

```
$varSqrt = sqrt(81);
echo $varSqrt;
```

在例 5.10 中，我们使用 sqrt()函数计算了 81 的平方根，并将计算结果赋值给了变量$varSqrt，然后使用 echo 语句输出了该变量的值。在这里只用了一个简单的函数就完成了求平方根的任务。

在 PHP 语言中，像这样的函数有很多，大多数都能在平常编写脚本时用到，节省了大量的编码时间。再来看两个示例。

【例 5.11】 上取整和下取整。

```
1   $varCeiling = ceiling (6.52);
2   $varFloor = floor(6.52);
3   $varTotal = $varCeling +$varFloor;
4   echo $varTotal;
```

在 PHP 中，ceiling()和 floor()两个函数都是用来对浮点型数取整用的。前者为上取整函数，后者为下取整函数。所谓上取整是指将某浮点数的整数部分加 1 作为取整后的值，而取值是指保留该浮点数的整数部分作为取整后的值。

在例 5.11 中，我们使用这两个函数对 6.52 这个浮点数进行了取整。按照上取整和下取整的定义，可知$varCeiling 的值为 7，而$varFloor 的值为 6。脚本的第三行定义了变量$varTotal，并为其赋值$varCeiling 和$varFloor 之和。因此，脚本第四行输出的$varTotal 的值应该为 13。大家可以自行验证一下。

5.2.3 数字格式化

在前面两节里，我们使用 echo 命令输出的数字都很小。对于比较长的数字，我们一时半会就很难认出它的大小来。为了解决这个问题，人们便以每三位加一个逗号的形式把数字给隔开。这样隔开以后，再长的数字也变得易读起来。

在编写 PHP 脚本时，也可以让系统输出带这种逗号的数字来。方法也很简单，只要使用 number_format()函数就好了。不过这个 number_format()函数的参数比较多。具体格式如下：

```
number_format(number, decimals, "decimalsep","thousandsep")
```

这四个参数中只有第一个参数是必选的，其他参数可选。那么它们都代表着什么意思呢，我们一个一个来看。

- number：代表的就是需要格式化的数字，这个参数是必选的。
- decimals：指的是小数位数，这个参数是可选的。如果没有指定小数位数，则该数字就会被四舍五入为一个整数后输出。当然，如果指定了 thousandsep，那么 decimals 就是必选的了。
- decimalsep：指的就是用于分隔整数部分和小数部分的符号，这个参数是可选的。默认使用的符号为点（.）。当然，如果指定了 decimalsep，那么 thousandsep 也就成了必选参数了。
- thousandsep：指的就是用于分隔百位与千位的符号，这个参数也是可选的。默认使用的符号为逗号（,）。当然，如果指定了 thousandsep，那么 decimalsep 也就成了必选参数了。

现在来看一些例子。由于该函数的使用比较简单，下面就以表格的形式为大家展示一

下该函数的使用方法和可能出现的结果，如表 5-2 所示。

表 5-2　number_format()函数的参数使用及其输出

$number	格　　式	输 出 结 果
12321	number_format($number)	12,321
12321.66	number_format($number, 2)	12,321.66
12321.66	number_format($number)	12,322
12321.6	number_format($number, 3)	12,321.600
12321	number_format($number, 0, ".", ".")	12.321
12321.66	number_format($number, 2, ".", "")	12321.66

5.3　咬文嚼字——字符串型数据

按照之前的讲法，字符串型数据可以用来存储字符信息。那么，什么是字符呢？所谓字符，指的就是字母、数字和标点符号等一切可以打印在屏幕上的符号。那么字符串指的自然就是一连串的字符了。如果一个数字被存储为字符串型的数据时，它仅仅就是一个字符或字符串，不能再用于数学运算。比如，我们经常会把电话号码和身份证号码之类的信息保存为字符串型的数据，虽然它们大多数都是纯粹由数字构成的。我们可以通过以下的语句把变量定义成字符串类型的变量。

【例 5.12】 定义字符串型变量。

```
$phoneNo = "15946892256";
$authorizationCode = "A34279";
$activationCode = (string) $phoneNo.$authorizationCode;
echo "Your activation code is ".$activationCode.".";
```

在例 5.12 中，定义了一个变量用于存放电话号码，还定义了一个变量用于存放授权码，然后把两个变量连接在一起组成激活码，最后把激活码打印在屏幕上。最后运行的结果应该是如下的一句话：

```
You activation code is 15946892256A34279.
```

5.3.1　文字游戏

在本小节里，我会传授给大家几招比较常见且十分有用的用于处理字符串的招式。大家在处理字符串时可以用到它们。

第一招　倒背如流

在处理字符串时，有时会要求把一个字符串倒过来写。其实这是一种十分简单的加密方式。PHP 为我们提供了一个预置函数 strrev()。使用这个函数，就可以把任何字符串反转过来。

【例 5.13】 字符串反转函数。

```
$varString = "This is a fine day!";
$varReverse = strrev($varString);
echo $varReverse;
```

执行例 5.13 中的脚本，会得到什么结果呢？

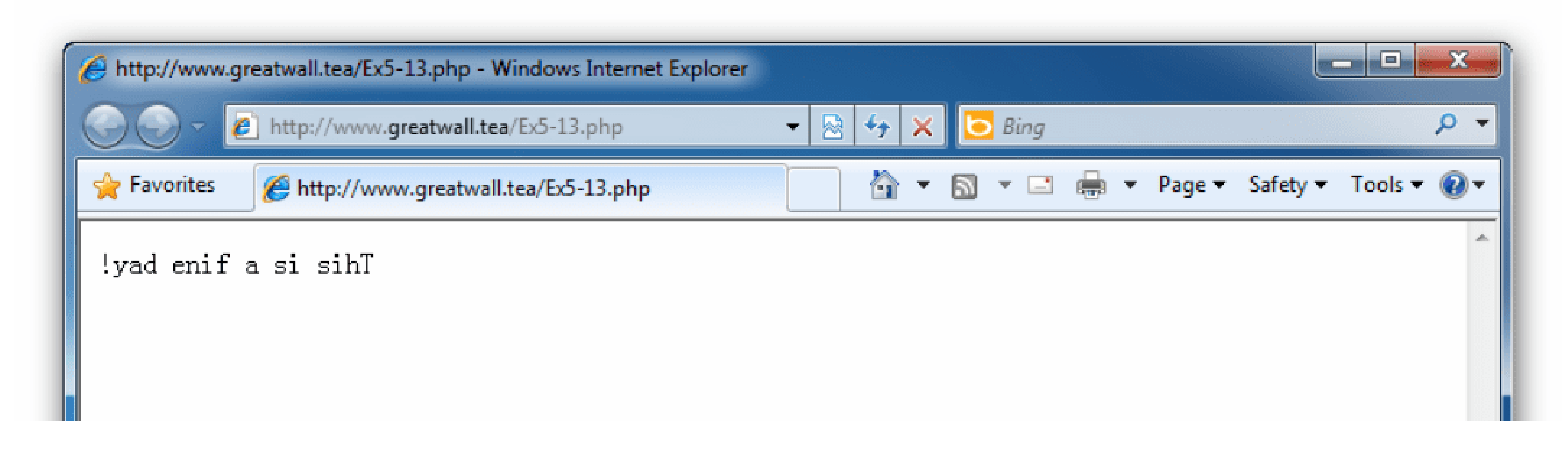


图 5-2　字符串反转函数

结果就是如图 5-2 所示，出现了一段看上去像是乱码的文字。再仔细辨认一下，才发现，原来这个函数会把字符串的最后的一个字符放到最前面，倒数第二个字符放到第二位，以此类推，实现反转的目的。这样倒背如流的功夫真是让人羡慕啊。

第二招　伸缩自如

有时候，我们希望把一个字符串中的英文字母全部变成大写的，有时候又希望把它们全部变成小写的，有时候我们希望让每个单词的首字母大写，有时候又希望让句子的第一个字母大写。我们的要求真是多的让人难以满足啊。不过 PHP 还是提供了一批函数来帮助我们达成愿望，让字符串们像孙悟空的金箍棒一样可以伸缩自如、忽大忽小啊。

【例 5.14】 字符串大小写函数。

```
$varSur = "windsor";
$varGiven = "william";
$varAffixTitle = "prince";
$varSuffixTitle = "duke of cambridge";
echo ucfirst($varAffixTitle)." ".ucfirst($varGiven)." ".strtoupper
($varSur).", ".ucwords($varSuffixTitle)."<br>";
```

这段脚本的执行结果如图 5-3 所示。

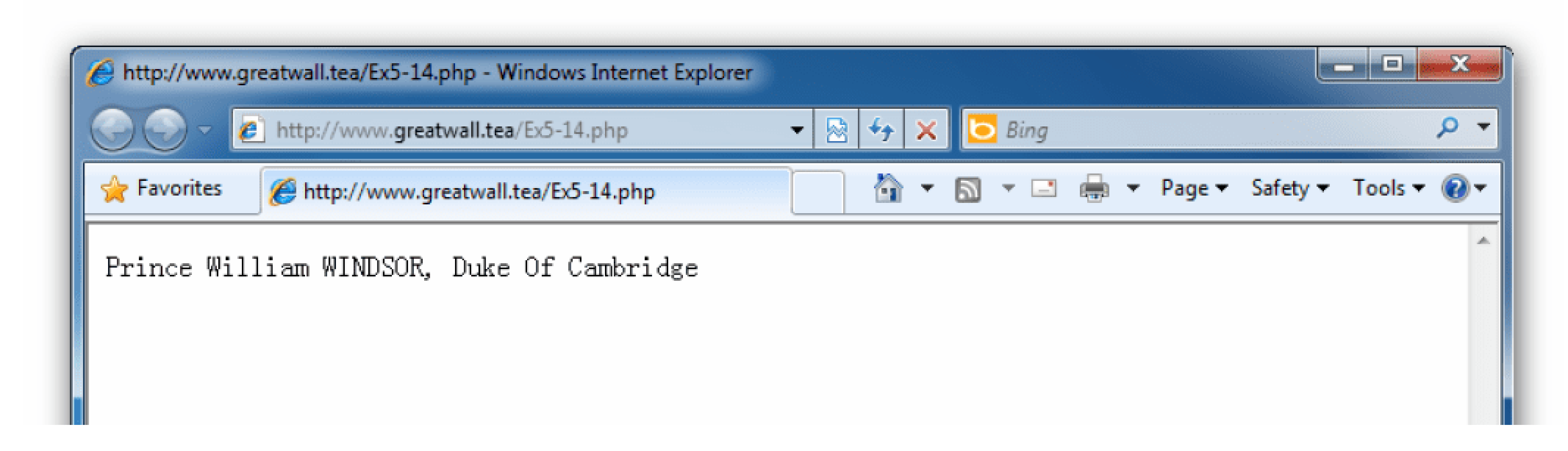


图 5-3　字符串大小写函数

在例 5.14 所示的脚本中，定义了三个函数，然后分别为它们赋值。大家可以注意到，用来赋值的字符串中所有的英文字母都是小写的。在输出的时候，使用了三个函数，它们分别是 ucfirst()、ucwords()和 strtoupper()。现在对照脚本的执行结果来看看这三个函数的功能。

在上述脚本里，

- ❑ 函数 ucfirst()中使用两次，代入的变量分别是$varAffixTitle 和$varGiven，显示的结果为首字母大写。
- ❑ 函数 ucwords()使用了一次，代入的变量为$varSuffixTitle，显示的结果为字符串的每个单词首字母大写。
- ❑ 函数 strtoupper()使用了一次，代入的变量为$varSur，显示的结果为字符串每个字母都大写。

其实，PHP 还提供了一个函数用来把字符串中的所有英文字母都变成小写的，这个函数就是 strtolower()。

这些函数名称看上去似乎没有什么规律，应该怎么记呢？如果你懂点英文，其实很好记：

- ❑ ucfirst 就是 upper case first 的缩写，意思是字符串首字母大写；
- ❑ ucwords 就是 upper case words 的缩写，注意 words 是复数形式，意思是字符串每个单词的首字母大写；
- ❑ strtoupper 就是 string to upper 的缩写，意思是字符串所有的字母都大写；
- ❑ strtolower 就是 string to lower 的缩写，意思是字符串所有的字母都小写。

第三招　断章取义

虽然我们平素里对断章取义这种行为深恶痛绝，但该断章取义的时候，我们还得断章取义不是？PHP 提供了这么一个函数，它就是 substr()。它有三个参数，分别是字符串、起始位置和子串长度。子串长度可以不指定，这样的话，截取的子串包含的就是从指定位置到字符串结尾的所有字符了。我们还是先来看一段脚本。

【例 5.15】 字符串截取函数（一）。

```
$varString = "Hello, this is my World!";
$firstSub = substr($varString, 0, 5);
$secondSub = substr($varString, 18);
echo $firstSub." ".$secondSub;
```

在例 5.15 这段脚本中，定义了变量$varString，并为其赋值“Hello, this is my World!”。然后，再使用 substr()函数截取了变量$varString 的部分内容定义了两个变量$firstSub 和$secondSub，在这里强调一下，substr()函数三个参数分别代表变量名、起始位置和子串长度。还要强调一下，函数 substr()是从第 0 位开始记录字符串中每个字符的位置的，也就是说字符串的第一个字符所在位置为 0，第二个字符所在位置才是 1。按照这个说明，大家可以自己推演一下脚本的运行结果。

如果读者足够心细，数数没有问题的话，应该会得到如图 5-4 所示的结果。

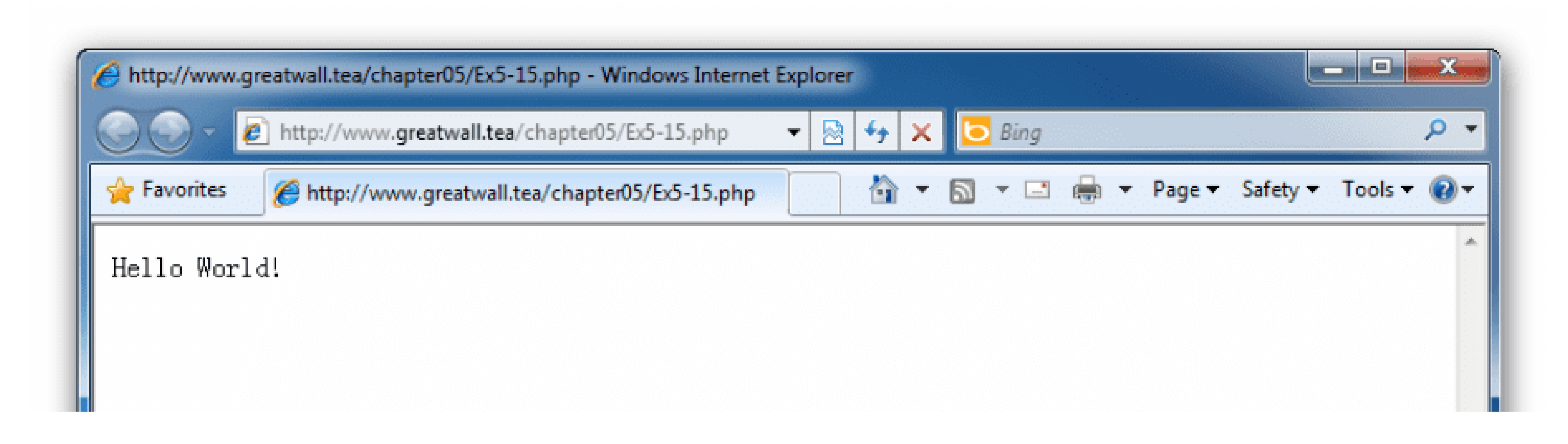


图 5-4　字符串截取函数

有的同学可能会说了，我还得靠数数才能确定某个字符所在的位置，如果字符串很长的话，那岂不是数数都要数吐血了？

其实没有那么夸张，如果实在不想这么复杂的话，还有变通的方法。这里我们把如例 5.15 所示脚本的第三行修改成如下的样子，运行的结果还是一样的。

```
3   $secondSub = substr($varString, -6);
```

上面这一句脚本是从字符串的末端开始数数，每个字符串最后一个字符的位置为第-1 位。我们需要截取“World!”这个子串，数数看，应该是从第-6 位起到字符串末尾止。

在这里，顺便再说一下，如果子串长度为负 N（N 为整数）的话，表明截取的子串包含从指定位置起到字符串末尾倒数第 N+1 位的所有字符。再来看一段脚本：

【例 5.16】 字符串截取函数（二）。

```
$varString = "Hello, this is my World!";
$firstSub = substr($varString, 4, 1);
$secondSub = substr($varString, -9, -1);
echo $firstSub.", ".$secondSub;
```

这段脚本的结果应该是：

```
o, my World
```

你数对了吗？

5.3.2　文本格式化

在 5.2.3 小节里，我们知道了如何格式化数字。在本小节里，我们将要学习如何格式化文本，也就是让文本以我们指定的格式输出。对于文本来说，定义输出格式是一件非常复杂的事情，就像在 Office 的 Word 组件中输出文字是十分容易的事情，但是让一个不太懂如何使用 Word 工具栏对输出的文本进行格式的排版却通常比写这些文字耗费的时间还要多。

中国人有一句话说的好：难者不会，会者不难。只要掌握了技巧，定义好文本的输出格式将是一件十分容易的事情。

在本小节里，将介绍两个以指定格式输出指定变量的函数，它们分别是 printf()和 sprintf()。前者相当于 echo，可以直接输出内容；后者则像 number_format()，只不过功能更加强大。它不光可以对文本进行格式化输出，对于数字也一样操控自如。这两个函数的参数如下：

```
printf("format", $varName1, $varName2, …);
$newVar = sprint("format", $varName1, $varName2, …);
```

来看下面一段脚本。

【例 5.17】 格式化文本（一）。

```
$nboys = 3;
$ngirls = 5;
printf("%s boys and %s girls", $nboys, $ngirls);
```

运行这段脚本后，显示的结果如图 5-5 所示。

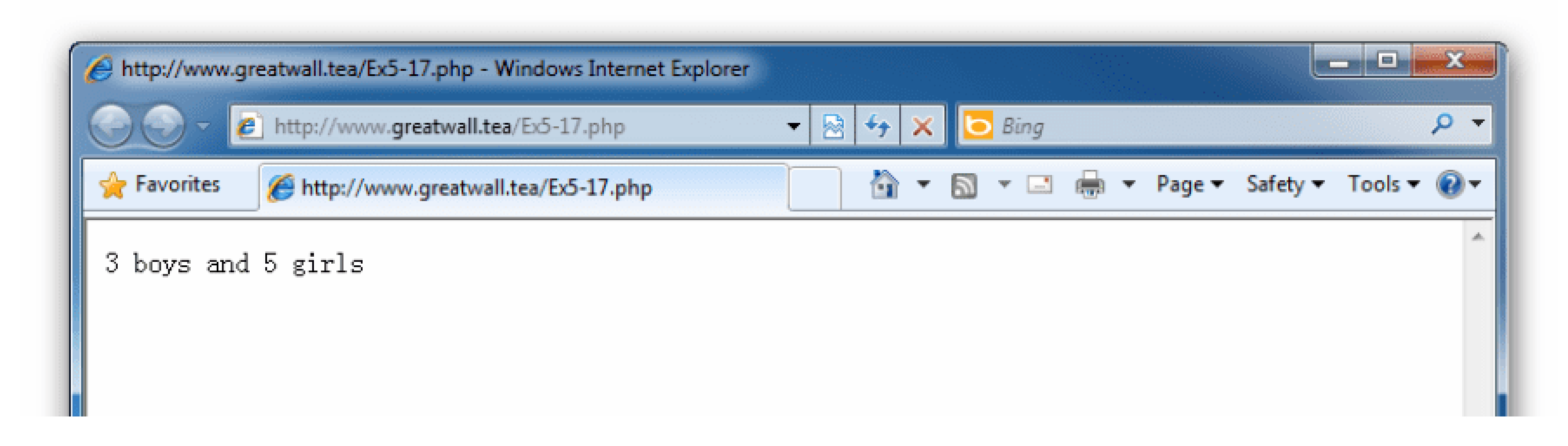


图 5-5　函数 printf()的输出结果

在这段脚本中，有一个特殊的符号“%s”，这就是定义文本输出格式的符号。就是这个“%s”告诉 PHP 处理引擎将后续指定的变量的值作为字符串按顺序插入到指定的位置。其中“%”告诉 PHP 处理引擎从这里开始执行文本格式化操作。其实这个指令的真实模样如图 5-6 所示。

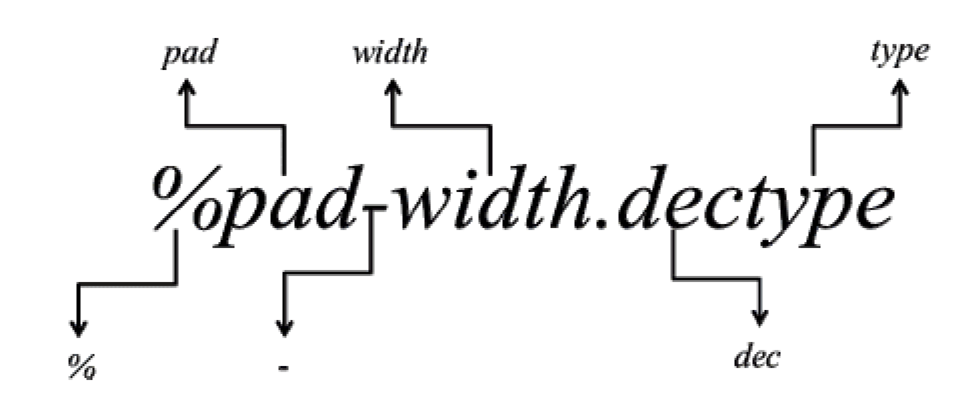


图 5-6　文本格式化指令

按照图 5-6 所示，这个指令由六个部分组成。下面分别对这六个部分进行说明：

- % （起始符），用于告诉 PHP 处理引擎从这里开始执行文本格式化操作。
- pad（占位符），用于在输出的文本长度小于指定长度时补齐两者之间的差距时所使用的占位符。比如某待输出变量的值的长度为 4，而指定长度为 8，那么输出的文本长度为 8，但其中 4 位为占位符。该占位符可以是 0、空格或任意字符。但需要注意的是，如果该占位符为字符，那么它的前面应该有一个单引号（'）。
- -（对齐符），用于告诉 PHP 处理引擎如何处理待输出某变量的值与占位符的左右关系。若使用了对齐符，待输出变量左对齐，即待输出变量在占位符的左边。若省略了对齐符，则待输出变量右对齐，即待输出变量在占位符的右边。还是用之前的例子，某待输出变量的值长度为 4 而指定的长度为 8，那么输出的文本长度为 8，但其中 4 位为占位符。若使用了对齐符，则待输出变量的值在 4 位占位符左边输出；若未使用对齐符，则待输出变量的值在 4 位占位符右边输出。
- width（指定长度），用于告诉 PHP 处理引擎输出的字符串长度。若待输出变量的值的长度与该处指定的长度有差距，则使用之前指定的占位符来补齐。比如，待输出变量的值为 1，指定长度为 5，占位符为 0，未使用对齐符，则输出的文本为 00001。
- .dec（小数位数），用于告诉 PHP 处理引擎在输出整形变量或浮点型变量时需要保留的小数位数。一定要注意保留前面的点（.）。
- type（变量类型），有两种类型可选：字符串型，用“s”表示；浮点型，用“f”表示。

那在例 5.17 中，我们看到的“%s”其实就是告诉 PHP 处理引擎以字符串的形式输出

指定变量的值。为了巩固上面讲到的知识，我们再来看一段脚本。

【例 5.18】 格式化文本（二）。

```
$cost1 = 456;
$commodity1 = "Microwave Oven";
$cost2 = 32.9;
$commodity2 = "Men's Underware";
$todayDate = "Jan. 21st, 2012";
$currentTime = "19:38:46";
$timeStamp = sprintf("Welcome to Goodbuy <br> %-25s%s <br>", $todayDate, $currentTime);
$billEntry1 = sprintf("%'.-25s%5.2f <br>", $commodity1, $cost1);
$billEntry2 = sprintf("%'.-25s%5.2f <br>", $commodity2, $cost2);
echo $timeStamp;
echo $billEntry1;
echo $billEntry2;
```

这段脚本输出的内容如图 5-7 所示。

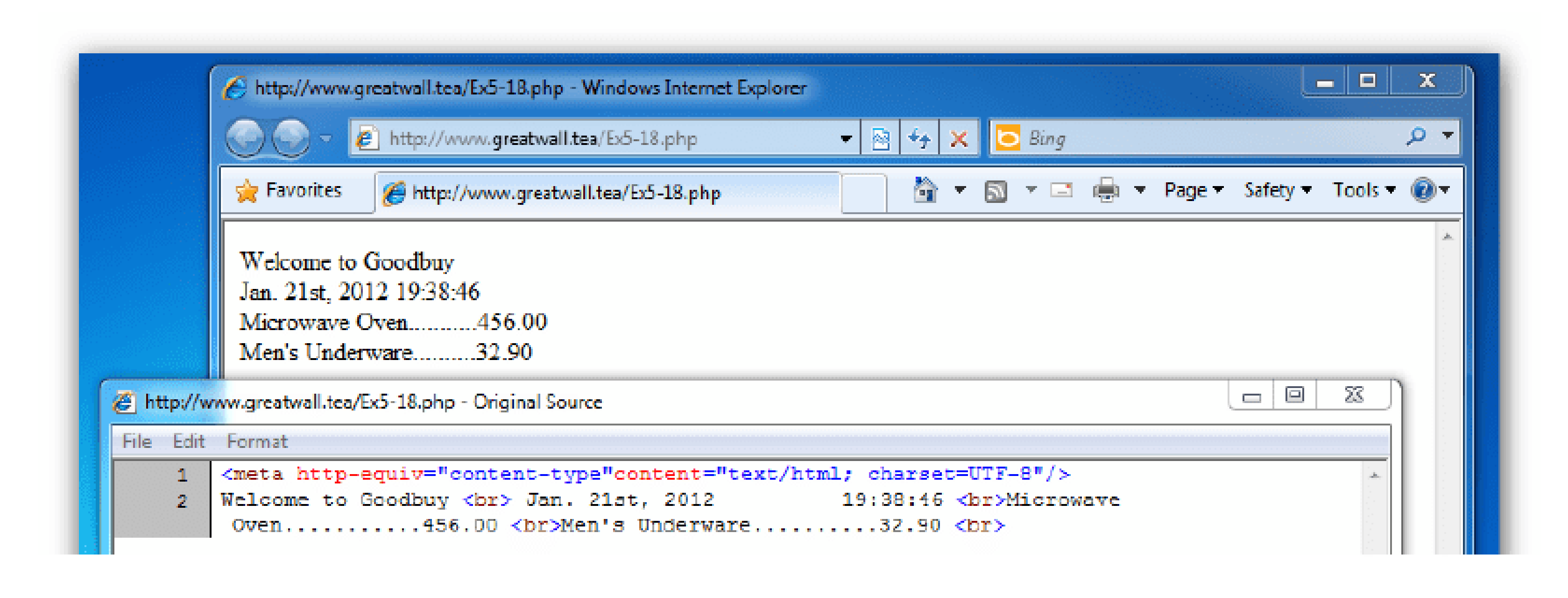


图 5-7　文本格式化（二）

运行例 5.18 中的脚本，我们打印了一张超市的小票。在脚本中，使用 sprintf()函数定义了三个段落。第一个段落为超市欢迎辞和小票打印的日期加时间，第二个段落为第一件商品的名称和价格，第三个段落为第二件商品的名称和价格。

在这三次使用 sprinf()函数的过程中，我们向 PHP 处理引擎下达了四个格式指令，它们分别是“%-25s”、“ %s”、“ %'.-25s”和“%5.2f”。第一个指令要求输出一串长度为 25 的字符串，该字符串包含以左对齐方式输出的指定变量的值和作为占位符的空格。第二个指令要求将指定变量的值以字符串的形式输出。第三个指令要求输出一串长度为 25 的字符串，该字符串包含以左对齐方式输出的指定变量的值和作为占位符的点（.）。第四个指令则要求将指定变量的值以字符串的形式输出，字符串的长度为 5，小数后保留 2 位。

通过图 5-7 所示的内容，可以发现，网页上两条消费记录的前后都是对齐的，而时间日期那一段前后并没有和下面的两条消费记录对齐。但是通过观察源代码，我们可以发现三个段落的长度都是一样的，均为 25 个字符。也就是说我们用作占位符的空格被浏览器给吃掉了。

5.4　操控时间——时间型数据

时间和日期在脚本里可以发挥非常大的作用。读者可以使用脚本进行基于时间的计

算，尤其是当需要给分布在世界各地的用户提供当地的时间信息时，时区换算成了最基本的要求。在计算机的世界里，时间是以时间戳的形式存储的。也就是说，所有的时间信息都是以秒为单位进行记录的。为了将时间戳里记录的信息转换成对人友好的格式，PHP 提供了许多函数。

在本节里，我们将通过了解这些函数的使用，来学习如何使用 PHP 脚本操控时间。

5.4.1　时间格式记

每个国家都有着自己约定俗成的时间格式。大家如果使用的是 Windows 的操作系统，可以在“控制面板”中找到“地区和语言”图标。双击后，可以看到如图 5-8 所示的“地区和语言”对话框。

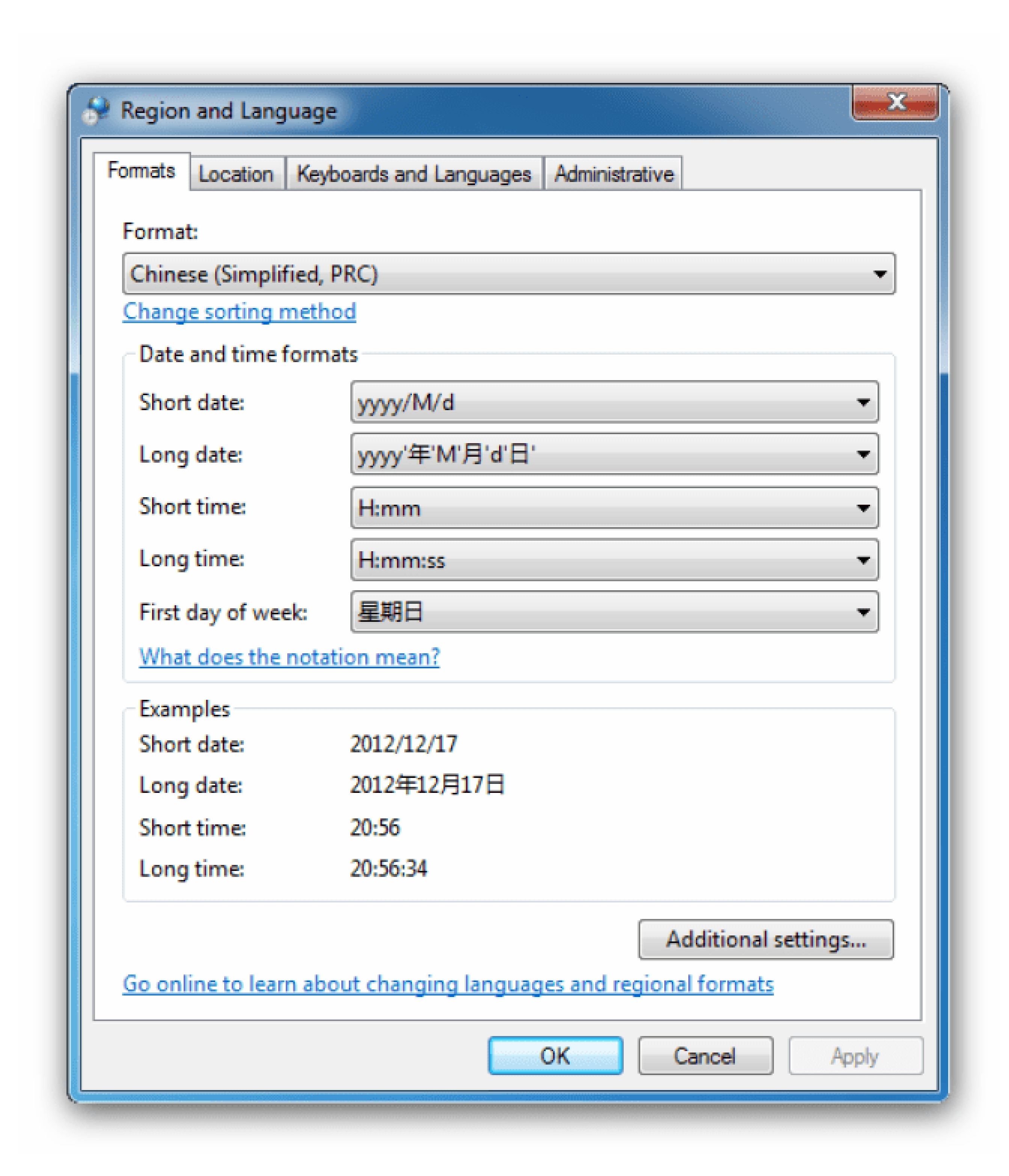


图 5-8　Windows 操作系统中的地区与语言对话框

在“格式”选项卡中，可以通过选择“格式”下拉列表框内的不同国家和地区来查看各国、各地区在书写时间和日期的时遵循的格式。

在 PHP 脚本中，经常使用的与时间和日期有关的函数就是 date()。这个函数可以将存储在时间戳里的时间信息按照我们指定的格式显示出来。这个函数格式如下：

```
date ("format", $timestamp);
```

先来看一段脚本。

【例 5.19】　使用 date()函数。

```
$today = date("Y/m/d");
```

```
2    echo "Today is ".$today.".";
```

假设今天是 2012 年 12 月 17 日，运行脚本后显示的结果如图 5-9 所示。

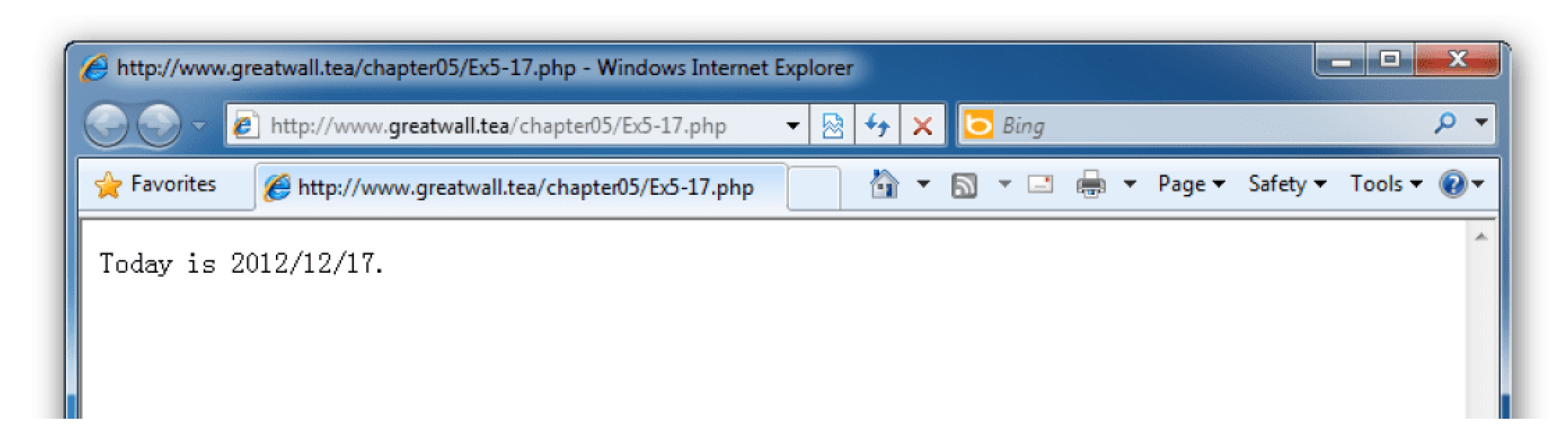


图 5-9　使用 date()函数

在例 5.19 中，只指定了 date()函数两个参数中的一个，并没有指定第二个参数。如果没有指定第二个参数，PHP 处理引擎会直接引用系统当前时间。现在我们来重点关注一下脚本中的第一个参数“Y/m/d”。为什么 Y 大写，而 m 和 d 都小写了呢？其实这是 PHP 用来区分时间格式的方法。比如“y-m-d”输出的是 12-12-17，而“M.d.Y”输出的是 Dec.17.12。表 5-3 列出了我们在定义时间格式时经常用到的一些符号。

表 5-3　时间与日期格式符

符号	意　　义	举　　例
M	月份的英文单词缩略语	Dec
F	月份的英文单词全拼	December
m	月份的数字形式，十月前的月份含前置的 0	02
n	月份的数字形式，十月前的月份不含前置的 0	2
d	日期的数字形式，十日前的日期含前置的 0	02
j	日期的数字形式，十日前的日期不含前置的 0	2
I	一星期中的某一天对应英文单词的全拼	Friday
D	一星期中的某一天对应英文单词的缩写	Fri
w	一星期中的某一天对应的数字，从 0（星期天）到 6（星期六）	5
Y	年份，四位数字	2012
y	年份，两位数字	12
g	小时，取值在 0 到 12 之间，十位以下无前置的 0	2
G	小时，取值在 0 到 24 之间，十位以下无前置的 0	2
h	小时，取值在 0 到 12 之间，十位以下有前置的 0	02
H	小时，取值在 0 到 24 之间，十位以下有前置的 0	02
i	分钟	00
s	秒钟	00
a	上午（am）或下午（pm）	am/pm
A	上午（AM）或下午（PM）	AM/PM
U	Unix 系统计秒	1056244941

5.4.2　时间型变量

在例 5.19 中，定义了一个变量，这个变量是时间型变量吗？答案自然是否定的，那个

变量是个字符串型变量。如果需要定义一个时间型的变量，首先得知道何为时间型变量。为此，我们先来看一段脚本。

【例 5.20】 获取时间戳。

```
$today = time();
echo $today;
```

在这一章的开始，提过时间戳这个概念。那么什么是时间戳呢？时间戳其实是计算机用来存储时间的方式。时间戳里存储的时间是以秒为单位的，记录的是指定时间与 1970 年 1 月 1 日 0 时 0 分 0 秒钟之间的时间差。在例 5.20 中，使用当前的时间定义了变量$today。然后输出这个变量，得到的应该会是如图 5-10 中的一段数字。

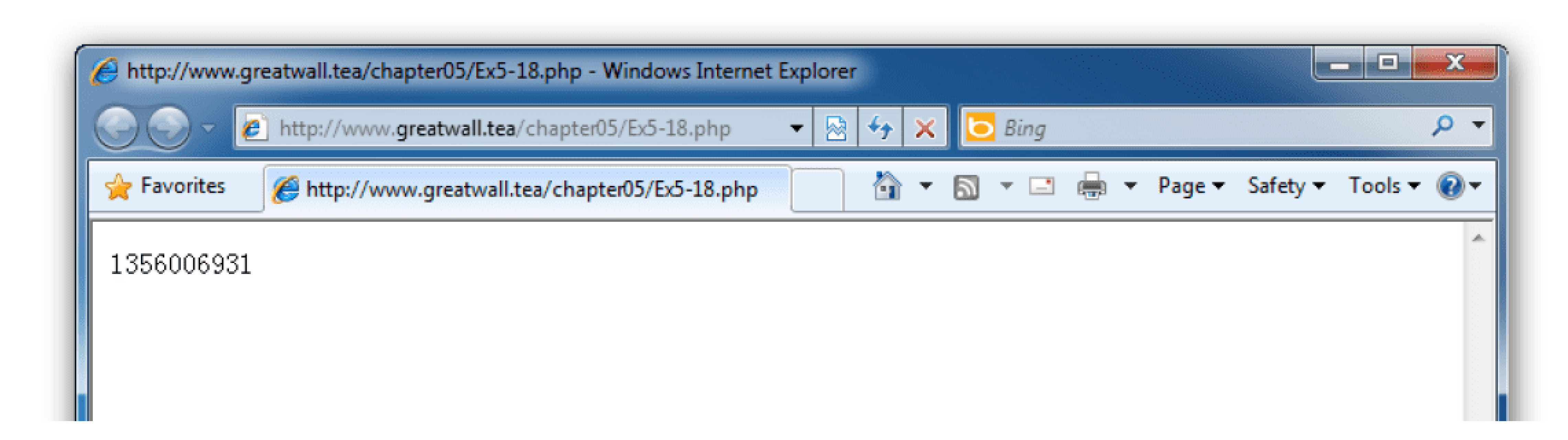


图 5-10　输出时间戳

当定义了一个时间型变量之后，就可以在上一节中提到的函数 date()中引用了。在 PHP 中，还可以通过其他方式，将指定时间以时间戳的形式存储到变量中。

【例 5.21】 存储指定时间到变量中（一）：strtotime()。

```
$importantDate = strtotime("today");        //输出当前时间的时间戳
$importantDate = strtotime("tomorrow");     //输出 24 小时后的时间戳
$importantDate = strtotime("last saturday");//输出上周六当前时间的时间戳
$importantDate = strtotime("now + 15 hours");  //输出 15 小时后的时间戳
$importantDate = strtotime("2 weeks ago"); //输出两周前当前时间的时间戳
$importantDate = strtotime("3 months later");
                                            //输出三个月后当前时间的时间戳
$importantDate = strtotime("next year gmt");
                               //以格林威治标准时输出明年今日当前时间的时间戳
$importantDate = strtotime("tomorrow 4am");
                               //输出明天凌晨四点当前时间的时间戳
```

大家一定觉得很神奇，引号里不就是一些英语短语么？居然可以这么用。其实 strtotime()函数认识的英文单词还有很多，在这里大致给分一下类。

- 月份名称：从一月到十二月的英文单词以及它们的缩写。
- 星期中的每一天：从周一到周日的英文单词以及它们的缩写。
- 时间单位：年（year）、月（month）、日（day）、星期（week）、小时（hour）、分（minute）和秒（second）等。
- 常用表示时间关系的词语：之后（ago）、现在（now）、上个（last）、下个（next）、这个（this）、明天（tomorror）和昨天（yesterday）。
- 加减号。

- 所有的数字。
- 时区：如 gmt（格林威治时间）、pdt（太平洋时间）和 bjt（北京时间）等。

这两个函数都比较简单，没有太多复杂的参数，下面介绍的这个函数参数比较多。大家在使用的时候务必要仔细。

【例 5.22】 存储指定时间到变量中（二）：mktime()。

```
$importantDate = mktime(0, 0, 0, 1, 15, 2003);
echo $importantDate;
```

函数 mktime()一共有六个参数，分别为小时、分、秒、月、日和年。例 5.20 中存储在变量$importantDate 中的日期如果用 strtotime()函数的话，就可以写成如下的形式：

```
$importantDate = strtotime("January 15 2003");
```

5.5　判别真假——布尔型数据

布尔型数据分为两类。

（1）true：表示事实成立，为真，也可以写成 true。

（2）false：表示事实不成立，为假，也可以写成 false。

如果将两个变量进行比较，并根据判断的结果来执行某些操作的话，我们可以这么写，如例 5.23 所示。

【例 5.23】 比较变量大小。

```
1   <?php
2       $num1 = 2;
3       $num2 = 5;
4       if($num1 == $num2){            //判断变量$num1 和变量$num2 是否相等
5           echo '$num1 等于 $num2';
6       } else {
7           echo '$num1 不等于 $num2';
8       }
9   ?>
```

在脚本的第 4 行，使用“==”对两个变量的值进行的比较。由于变量$num1 和变量$num2 的值并不相等，因此“$num1 == $num2”的值为 false，也就是说变量$num1 和变量$num2 相等的判断不成立。这时，系统会输入第二句 echo 语句，也就是“$num1 不等于$num2”。

当然，其他类型的变量也可以转换成布尔型变量。如下的变量转换成布尔型变量时，其值为 false：

- 整型值 0（零），
- 浮点型值 0.0（零），
- 空白字符串和字符串 "0"，
- 没有成员变量的数组，
- 没有单元的对象（仅适用于 PHP 4），
- 特殊类型 NULL（包括尚未设定的变量）。

除上述这些变量之外，其他类型的变量在转换成布尔型变量之后，其值为 true。

5.6　实战练习：计算税后收入

5.6.1　背景介绍

在这一节里，我们编写一段脚本来帮助生活在北京地区的人们计算其税后收入。按照北京市政府的规定，城镇职工需按月缴纳住房公积金和社会保险，并在其当月的工资中扣除。扣除后剩余的工资如果超过个人所得税征税标准的，需要按照个人所得税征税标准征收个人所得税。表 5-4 所示列出了在计算个人所得税前需扣除的住房公积金和社会保险的比例。表 5-5 所示列出了国家现行的个人所得税征收比率。

表 5-4　北京地区城镇职工住房公积金和社会保险占月收入比例

		个人缴纳比例	用人单位缴纳比例
住房公积金		12%	12%
社会保险	养老保险	8%	20%
	医疗保险	2% + 3	10%
	工伤保险	0%	2%
	失业保险	0.2%	1%
	生育保险	0%	0.8%

表 5-5　国家现行个人所得税税率表

级数	全月应纳税所得额		税率(%)	速算扣除数
	含税级距	不含税级距		
1	不超过 1500 元的	不超过 1455 元的	3	0
2	超过 1500 元至 4500 元的部分	超过 1455 元至 4155 元的部分	10	105
3	超过 4500 元至 9000 元的部分	超过 4155 元至 7755 元的部分	20	555
4	超过 9000 元至 35000 元的部分	超过 7755 元至 27255 元的部分	25	1005
5	超过 35000 元至 55000 元的部分	超过 27255 元至 41255 元的部分	30	2755
6	超过 55000 元至 80000 元的部分	超过 41255 元至 57505 元的部分	35	5505
7	超过 80000 元的部分	超过 57505 元的部分	45	13505

在这两张表中，只需要关注底色为灰色的部分就可以了。现在我们开始搭建税后收入计算器。同前两章的实战练习一样，先来搭建用户界面：

在图 5-11 的左侧有一个表单，供用户输入姓名、性别和税前收入。当用户填好表单，并单击“计算税后收入”按钮后，在表单的右侧会出现一张报表。其中详细地列出了该用户应该缴纳的五险一金和个人所得税的金额。

我们一起来看看这是如何实现的吧。

5.6.2　实现过程

先来看 HTML 部分：

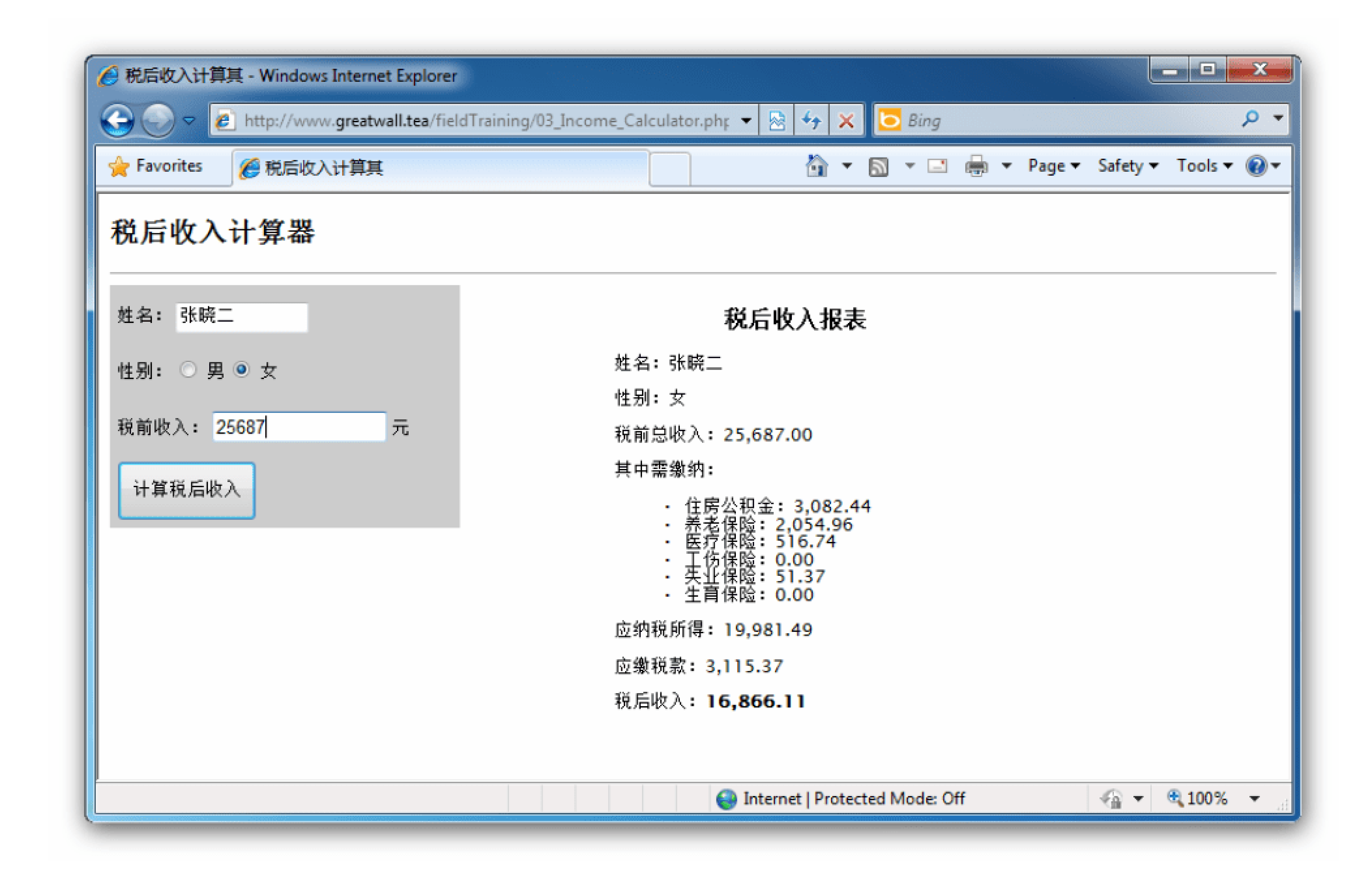


图 5-11　税后收入计算器用户界面

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>税后收入计算器</title>
        <style type="text/css">
            body {
                font-size:small;
            }

            form div {
                line-height:35px;
                margin:5px 5px 5px 5px
            }

            .form-container {
                width:30%;
                float: left;
                background-color:#cccccc;
            }

            .info-container {
                width:55%;
                float:left;

                margin-left:10px;
            }

            .info-container p, ul li {
                line-height:12px;
                margin-left:100px;
                font-family:"Lucida Sans Unicode";
            }

            .info-container h3 {
```

```
                text-align:center;
            }

            ul {
                margin-left: 10px;
            }

        </style>
    </head>
    <body>
        <h2>税后收入计算器</h2>
        <hr />
        <div class="form-container">
            <form action="?check" method="post">
                <div>
                    <label for="user">姓名：</label>
                    <input type="text" size="10" name="user">
                </div>
                <div>
                    <label for="gender">性别：</label>
                    <input type="radio" name="gender" value="male" checked>
                    男
                    <input type="radio" name="gender" value="female">
                    女
                </div>
                <div>
                    <label for="income">税前收入：</label>
                    <input type="text" size="15" name="income">
                    元
                </div>
                <div>
                    <button type="submit">计算税后收入</button>
                </div>
            </form>
        </div>
        <div class="info-container" id="info"></div>
                                    //用于存放用户在提交表单后产生的各项数据
    </body>
</html>
```

在这段 HTML 代码中，使用 CSS 和 DIV 定义了两个名为“form-container”和“info-container”的 DIV，分别用来存放表单和生成的纳税报表。与上一章的实战演练一样，表单的 action 属性的值定义为“=?check”，以便后续判断用户是否提交了表单。

再来看看 PHP 的部分。与前面两章一样，我们需要在“</body>”标签前添加 PHP 脚本。

```
<?php
    if(isset($_REQUEST['check'])){                  //判断用户是否提交了表单
        $usr = $_REQUEST['user'];                   //获取用户名
        $gen = $_REQUEST['gender'];                 //获取用户性别
        $inc = preg_match('/^[0-9]*$/', $_REQUEST['income']) ?
                intval($_REQUEST['income']) : '请输入您的税前收入。';
        /* 上面这是个三目表达式，问号前为判断条件，问号后以中间的冒号为界，
          当判断条件为真时，用冒号前的表达式为变量赋值；若判断条件为假，则
          用冒号后的表达式为变量赋值。                              */
        /* 在上面这个三目表达式中，我们使用了 preg_match()函数对用户输入的税前收入
```

```
    进行验证，关于 preg_match()这个函数的介绍，请见第 7 章。              */

if (is_int($inc)) {
        //由于变量$inc 的值可能为整数，也可能为字符串。因此需要判断一下
    $zhuf = $inc * 0.12;         //计算住房公积金
    $yang = $inc * 0.08;         //计算养老保险
    $yili = $inc * 0.02 + 3;     //计算医疗保险
    $gong = $inc * 0.00;         //计算工伤保险
    $shiy = $inc * 0.002;        //计算失业保险
    $shen = $inc * 0.00;         //计算生育保险

    $beforeTax = $inc - $zhuf - $yang - $yili - $gong - $shiy - $shen;
                                 //计算应税所得
    if($beforeTax > 3500 && $beforeTax <= 5000) {
        $tax = ($beforeTax-3500) * 0.03;
    } elseif ($beforeTax > 5000 && $beforeTax <= 8000) {
        $tax = ($beforeTax-3500) * 0.1 - 105;
    } elseif ($beforeTax > 8000 && $beforeTax <= 12500) {
        $tax = ($beforeTax-3500) * 0.2 - 555;
    } elseif ($beforeTax > 12500 && $beforeTax <= 38500) {
        $tax = ($beforeTax-3500) * 0.25 - 1005;
    } elseif ($beforeTax > 38500 && $beforeTax <= 58500) {
        $tax = ($beforeTax-3500) * 0.3 - 2755;
    } elseif ($beforeTax > 58500 && $beforeTax <= 83500) {
        $tax = ($beforeTax-3500) * 0.35 - 5505;
    } elseif ($beforeTax > 83500) {
        $tax = ($beforeTax-3500) * 0.4 - 13505;
    } else {
        $tax = 0;
    }                                        //计算应缴税款

    $afterTax = $beforeTax - $tax;           //计算税后收入

    $msg = '<h3>税后收入报表</h3>';
    $msg .= '<p>姓名：'.$usr.'</p>';
    if($gen == 'male') {
        $msg .= '<p>性别：男</p>';
    } else {
        $msg .= '<p>性别：女</p>';
    }
    $msg .= '<p>税前总收入：'.number_format($inc,2).'</p>';
                                             //格式化输出
    $msg .= '<p>其中需缴纳：</p>';
    $msg .= '<ul><li>住房公积金：'.number_format($zhuf,2).'</li>';
    $msg .= '<li>养老保险：'.number format($yang,2).'</li>';
    $msg .= '<li>医疗保险：'.number_format($yili,2).'</li>';
    $msg .= '<li>工伤保险：'.number_format($gong,2).'</li>';
    $msg .= '<li>失业保险：'.number_format($shiy,2).'</li>';
    $msg .= '<li>生育保险：'.number_format($shen,2).'</li></ul>';
    if($tax > 0){             //根据应缴税款是否为 0，向浏览器输出纳税提示
        $msg .= '<p>应纳税所得：'.number_format($beforeTax,2)
        .'</p>';
        $msg .= '<p>应缴税款：'.number_format($tax,2).'</p>';
        $msg .= '<p>税后收入：<strong>'.number_format
        ($afterTax,2).'</strong></p>';
```

```
            } else {
                $msg .= '<p>您无需缴纳个人所得税.</p>';
            }
        } else {
            $msg = $inc;
        }
        echo '<script language="javascript">document.getElementById
        ("info").innerHTML="'.
            $msg.'"</script>';  //向 HTML 中的 ID 为 info 的 div 输出税后收入报表
    }
?>
```

上面这段代码虽然比较长，但是总共也不过 6 个步骤。它们分别是：

（1）获取用户提交的信息；

（2）计算应税所得；

（3）计算应缴税款；

（4）计算税后收入；

（5）生成税后收入报表；

（6）将税后收入报表输出到页面的指定位置。

在这 6 个步骤中，第 1 步（如何获取用户提交的信息）和第 6 步（如何将生成的信息通过 JavaScript 输出到浏览器中）与第 4 章实战演练的内容相同。如果大家看着代码也想不起来如何实现这两个步骤的话，可以返回第 4 章的实战演练部分再看一下。

在这里我们重点关注一下第 2 步到第 5 步的内容。

在第 2 步中，使用$_REQUEST 常量数组获取用户通过表单提交的三个元素的值，并将获取到的内容分别赋值给$usr、$gen 和$inc 三个变量。这三个变量分别代表着用户输入的姓名、性别和税前收入。其中，在为变量$inc 赋值时，我们使用了三目表达式。

所谓的三目表达式本来是微软开发的 C 语言中特有的一种表达式。由于 PHP 脚本语言继承了 C 语言的大部分语言风格，因此三目表达式在 PHP 中也是可以使用的。

在一个三目表达式中，一共有三个子表达式，分别为“判断表达式”、“判断为真时的赋值表达式”及“判断为假时的赋值表达式”。这三个子表达式之间用“？”和“：”分隔开，格式如下：

```
$var = expression 1 ? expression 2 : expression 3
```

使用这个表达式为变量$var 赋值时，若“expression 1”的值为真，则将“expression 2”的值赋给变量$var，若“expression 2”的值为假时，则将“expression 3”的值赋给变量$var。

脚本中为变量$inc 赋值使用的三目表达式如下：

```
$inc = preg_match('/^[0-9]*$/', $_REQUEST['income']) ?
        intval($_REQUEST['income']) : '请输入您的税前收入。';
```

在这个表达式中，我们使用 preg_match()函数来判断用户在表单的“税前收入”文本框中输入的是不是一个大于零的正整数。若是则将该值的数据类型由字符串型转换成整型；若不是则将变量$inc 的值定义为“请输入您的税前收入”。

有的同学看到这里可能会问了：为什么不用大于和小于号来判断用户输入的值呢？要知道系统通过表单传递的值，其类型都是字符串，为了避免在计算过程中因数据类型不一致而发生错误，我们最好使用正则表达式来对页面间传递的数据进行验证。关于正则表达

式，大家可以看看第 7 章的相关内容，在这里就不再赘述了。

由于这个三目表达式，变量$inc 可能为一个整型变量，也可能为一个字符串型变量。基于此，我们使用了一个 if 语句用来判断变量$inc 的数据类型。若变量$inc 为一个整型变量，则继续执行第 3 步至第 5 步的内容。若变量$inc 为一个字符串型变量，则略过第 3 步至第 5 步的内容，直接向浏览器输出错误提示，要求用户输入正确的税前收入。

有了对数据类型的判断，我们就可以保证在执行第 3 步到第 5 步时，不会因数据类型的问题出现错误了。

在第 3 步里，我们按照表 5-4 中的规定先计算了在税前应该扣除的五险一金，并将这些数据存储在名为$zhuf、$yang、$yili、$gong、$shiy 和$shen 六个变量中，分别代表住房公积金、养老保险、医疗保险、工伤保险、失业保险和生育保险。然后，我们从税前收入减去这六个变量的值得到应税所得，并将应税所得赋值给变量$beforeTax。

在第 4 步里，我们根据计算得出的应税所得和表 5-5 的内容计算应缴税款。按照现行的个人所得税税率表，可以很方便的计算出不同档收入对应的应缴税款。公式如下：

应缴税款 ＝ （应税所得–所得税起征点）×相应档位的税率 – 同档速算扣除数

举个例子，若某先生的应税所得为 10,000 元，现行个人所得税的起征点为 3500 元。则该先生应该按照第三档标准纳税。该档位的税率为 0.20，速算扣除数为 555 元。因此，该先生的应缴税款为 754 元。

按照这个公式，我们使用 if…else 语句在脚本中列出了七个档位的计算公式及使用这些公式的条件。系统会根据第 3 步中计算出的应税所得自动套用这七个档位的计算公式中的一个来计算应缴税款。

在计算去应缴税款后，我们用第 3 步中计算出的应税所得减去应缴税后，即可得到税后收入。

在第 5 步里，我们开始输出计算结果到报表中。由于此处比较复杂，请允许我在下方再次列出输出报表的脚本。

```
$msg = '<h3>税后收入报表</h3>';
$msg .= '<p>姓名：'.$usr.'</p>';
if($gen == 'male') {
    $msg .= '<p>性别：男</p>';
} else {
    $msg .= '<p>性别：女</p>';
}
$msg .= '<p>税前总收入：'.number_format($inc,2).'</p>';      //
$msg .= '<p>其中需缴纳：</p>';
$msg .= '<ul><li>住房公积金：'.number_format($zhuf,2).'</li>';
$msg .= '<li>养老保险：'.number_format($yang,2).'</li>';
$msg .= '<li>医疗保险：'.number_format($yili,2).'</li>';
$msg .= '<li>工伤保险：'.number_format($gong,2).'</li>';
$msg .= '<li>失业保险：'.number_format($shiy,2).'</li>';
$msg .= '<li>生育保险：'.number_format($shen,2).'</li></ul>';
if($tax > 0){                    //根据应缴税款是否为 0，向浏览器输出纳税提示
    $msg .= '<p>应纳税所得：'.number_format($beforeTax,2).'</p>';
    $msg .= '<p>应缴税款：'.number_format($tax,2).'</p>';
    $msg .= '<p>税后收入：<strong>'.number_format($afterTax,2)
    .'</strong></p>';
```

```
} else {
    $msg .= '<p>您无需缴纳个人所得税.</p>';
}
```

在第 5 步的脚本里，先定义了一个变量$msg，并将其值定义为“<h3>税后收入报表</h3>”。值得注意的是，在这段脚本里，除了在定义变量$msg 时使用的是赋值符（=）之外，其他使用变量$msg 的地方都在赋值符前加了一个点（.），表示将“.=”后的内容附加到变量$msg 中。这个点其实就是我们在 5.3 节里学到的连接字符串的运算符。

在定义了变量$msg 后，我们将获取用户输入的“姓名”和“性别”附加到变量中，接着把我们计算得出的“税前收入”和“五险一金”附加到变量中。最后根据应缴税款是否为零，来决定是输出“应税所得”、“应缴税款”和“税后收入”，还是输出“您无需缴纳个人所得税”这样一句提示。

在向变量$msg 中附加这些信息时，使用了一些 HTML 代码对这些信息进行了格式化，使得这些信息可以如图 5-11 所示的那样有序输出。

至此，我们就完成了税后收入计算器脚本的编写。

5.7　习　　题

（1）请判断如下脚本的运行结果：

```
<?php
    $a = (int) 5.6;
    $b = (float) 5.6;
    $c = $a + $b;
    echo $c;
?>
```

（2）请使用 sprint()函数输出本章的目录，注意一级标题与二级标题之间、以及二级标题与三级标题之间有缩进。

（3）请使用 strtotime()函数和 date()函数输出当前日期前 25 天的日期、前一个月的日期、前五周的日期、前三个月的日期和前半年的日期。

（4）请使用三目表达式编写一段脚本。

第 6 章　抱团效应——数组

数组？什么数组？看到这个词，很多同学的第一反应就是这个。难不成是一组数吗？其实，还真可以把数组看成是用于存放一组数据的复杂变量。比如，可以把关于用户的一组信息存放在这个变量$userInfo 下。存放完成后，可以很方便地读取、修改和处理数组中存放的数据。

在本章里，我们将学习如何创建、修改、复制和使用数组。

6.1　多胎胞——数组的声明与使用

在 PHP 的概念里，数组有着十分重要的作用。本节将学习如何创建、修改和删除数组。

6.1.1　创建数组

如同通过给变量赋值来创建变量一样，可以通过给变量赋值一组数组来创建数组。例如，可以使用下面的语句来定义一个名为$userInfo 的数组：

```
$userInfo[1] = "Jim Green";
```

通过这句脚本，一个名为$userInfo 的数组就创建好了。在这个数组中，有一个元素，其值为 Jim Green。接下来，可以使用下面的语句为这个数组添加新的元素：

```
$userInfo[2] = "18";
$userInfo[3] = "Male";
```

现在，$userInfo 就是一个拥有三个元素的数组了。

按照这样的说法，一个数组可以被看做是一个键值对（key-value pair）的列表，信息将按照以下的结构被存储起来：

```
$arrayName['key1'] = value1;
$arrayName['key2'] = value2;
$arrayName['key3'] = value3;
…
```

一个数组可以包含无限个元素。这里的键（key）又可以被称为索引值。在很多的编程语言中，数组的索引只能是数字形式的，而在 PHP 等少数几种脚本语言和编程语言中，数组的索引既可以是数字形式的，也可以是字符串形式的。为了说明这个问题，我们使用下面的语句来重新定义一下$userInfo 这个数组：

```
$userInfo["Name"] = "Jim Green";
$userInfo["Age"] = "18";
```

```
$userInfo["Gender"] = "Male";
```

这样一来，我们就建立了索引和值之间一一对应的关系，并且也便于其他程序员阅读这段脚本。当然，如果像一个用来储存价格的数组，可以使用更为快捷的方式定义它：

```
$price[] = 19.25;
$price[] = 22.35;
$price[] = 133.24;
```

在定义数组$price 这个数组时，我们没有指定索引值。当 PHP 处理引擎在处理到这样的数组时会自动为其分配数字形式的索引值。不过自动分配的索引值是从 0 开始的。也就是说，上面这段脚本等价于：

```
$price[0] = 19.25;
$price[1] = 22.35;
$price[2] = 133.24;
```

除了通过直接赋值的方式创建数组之外，我们还可以通过 array()函数来创建数组。例如，要创建一个名为$fruits 的数组，那么可以用下面的语句来声明这个数组：

```
$fruits = array("Apple", "Orange", "Banana", "Grapefruit");
```

那么$fruits[3]的值就是 Grapefruit 了。这也说明了，通过 array()函数创建的数组中各元素的索引值也是由系统从 0 开始自动分配的。那么，我们可不可以在使用 array()函数的同时，使用字符串形式的索引值呢？为了实现这个愿望，PHP 提供了如下的方式：

```
$arrayName = array('key1' => 'value1',
'key2' => 'value2',
'key3' => 'value3',
'key4' => 'value4');
```

在使用 array()函数时，我们使用“=>”符号来定义键值对。这个符号的左边是数组元素索引，而右边是元素值。每两个键值对之间用逗号隔开。为了方便脚本维护，也可以在用于隔开键值对的逗号后回车。这样做不会影响 PHP 处理引擎对脚本的处理。

除了可以在 array()函数中定义字符串形式的索引之外，还可以通过为第一个元素指定不为 0 的数字索引来为数组中的元素自动分配数字形式的索引，例如：

```
$topNations = array('12' => 'India', 'Pakistan','Israel','Budan');
```

通过这条语句定义的数组$topNations[2]的值为未定义，而$topNations[14]为 Israel。

另外，如果需要创建一个以连续值为元素的数组，则可以使用 range()函数来定义数组。比如：

```
$year = range(2002, 2012);
```

这条语句相当于：

```
$year[0] = 2002;
$year[1] = 2003;
…
$year[10] = 2012
```

类似的，我们还可以使用 range()来定义数组$alphabet：

```
$alphabet = range('a', 'z');
```

这条语句相当于：

```
$alphabet[0] = 'a';
$alphabet[1] = 'b';
$alphabet[2] = 'c';
…
$alphabet[24] = 'y';
$alphabet[25] = 'z';
```

6.1.2　查看数组

在创建了数组之后，可以使用 var_dump()函数和 print_r()函数来查看数组的结构。前者和后者的区别在于，前者提供了更为丰富的关于数组元素的信息，而后者与之相比略显简单。

现在，我们分别使用这两个函数来查看在上一节一开始就创建了的数组$userInfo 的结构，也就是如例 6.1 所示的脚本。

【例 6.1】 查看数组结构（一）。

```
<?php
    $userInfo[0] = "Jim Green";
    $userInfo[1] = 18;
    $userInfo[2] = "Male";

    echo "The following displays the output of the var_dump function:<br>";
    echo "<pre>";
    var_dump($userInfo);
    echo "</pre>";

    echo "<br>The following displays the output of the print_r function:<br>";
    echo "<pre>";
    print_r($userInfo);
    echo "</pre>";
?>
```

得到的结果如图 6-1 所示。

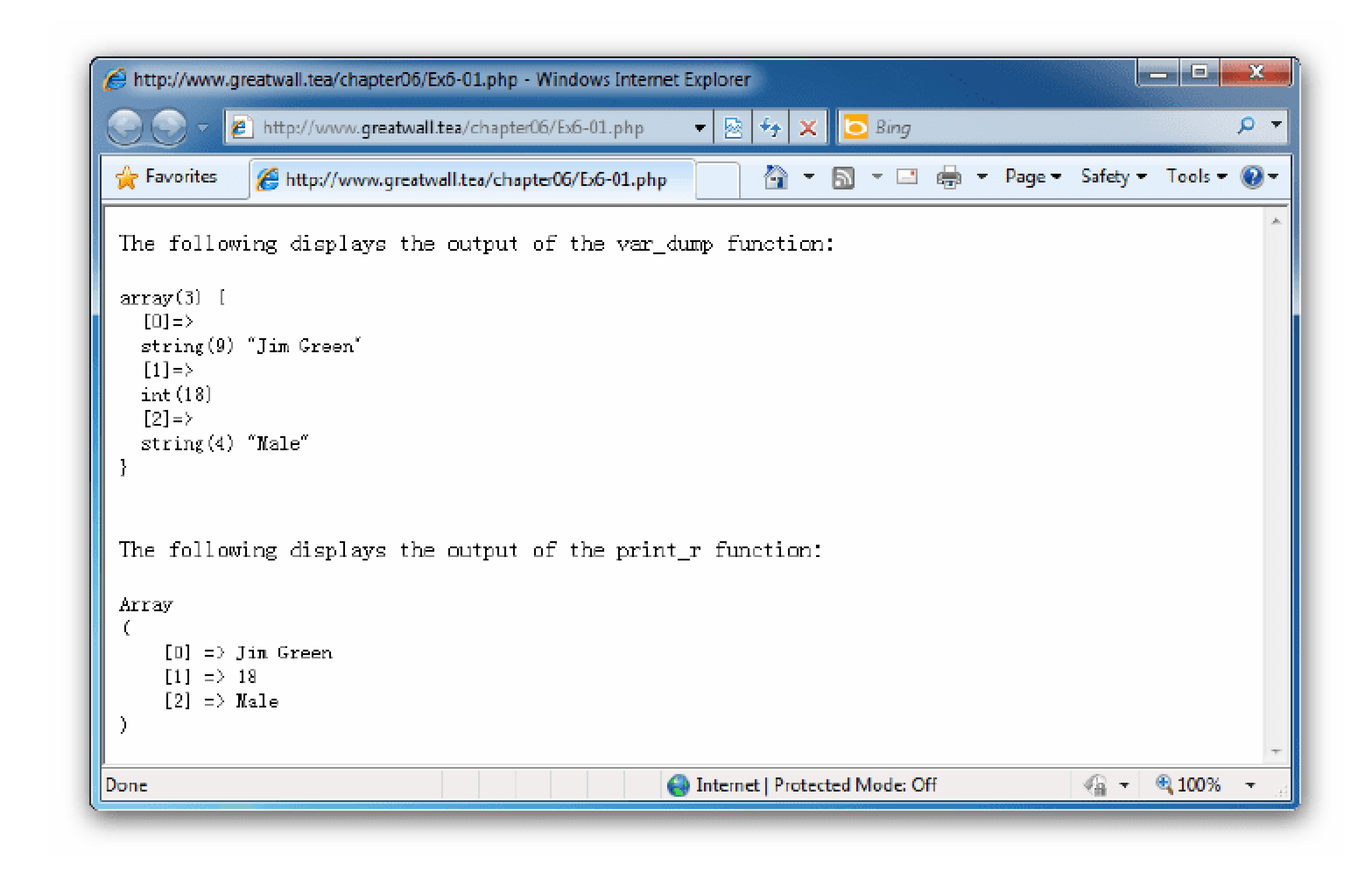


图 6-1　查看数组结构（一）

按照图 6-1 所示，var_dump()函数除了给出 print_r()函数给出的数组各元素键值对之外，还提供了如下信息：

- ❑ 数组元素个数；
- ❑ 数组元素的数据类型；
- ❑ 若某元素是字符串型的数据，还输出了该字符串的长度。

如果在数组中使用字符串形式的索引，那么在查看数据结构时，键值对中出现的索引就会被包含在一对引号和一对方括号之间，比如例 6.2 所示。

【例 6.2】 查看数组结构（二）。

```
<?php
    $userInfo["Name"] = "Jim Green";
    $userInfo["Age"] = 18;
    $userInfo["Gender"] = "Male";

    echo "The following displays the output of the var_dump
    function:<br>";
    echo "<pre>";
    var_dump($userInfo);
    echo "</pre>";

    echo "<br>The following displays the output of the print_r
    function:<br>";
    echo "<pre>";
    print_r($userInfo);
    echo "</pre>";
?>
```

在使用 var_dump()函数和 print_r()函数查看数组$userInfo 的结构时，显示的结构如图 6-2 所示。

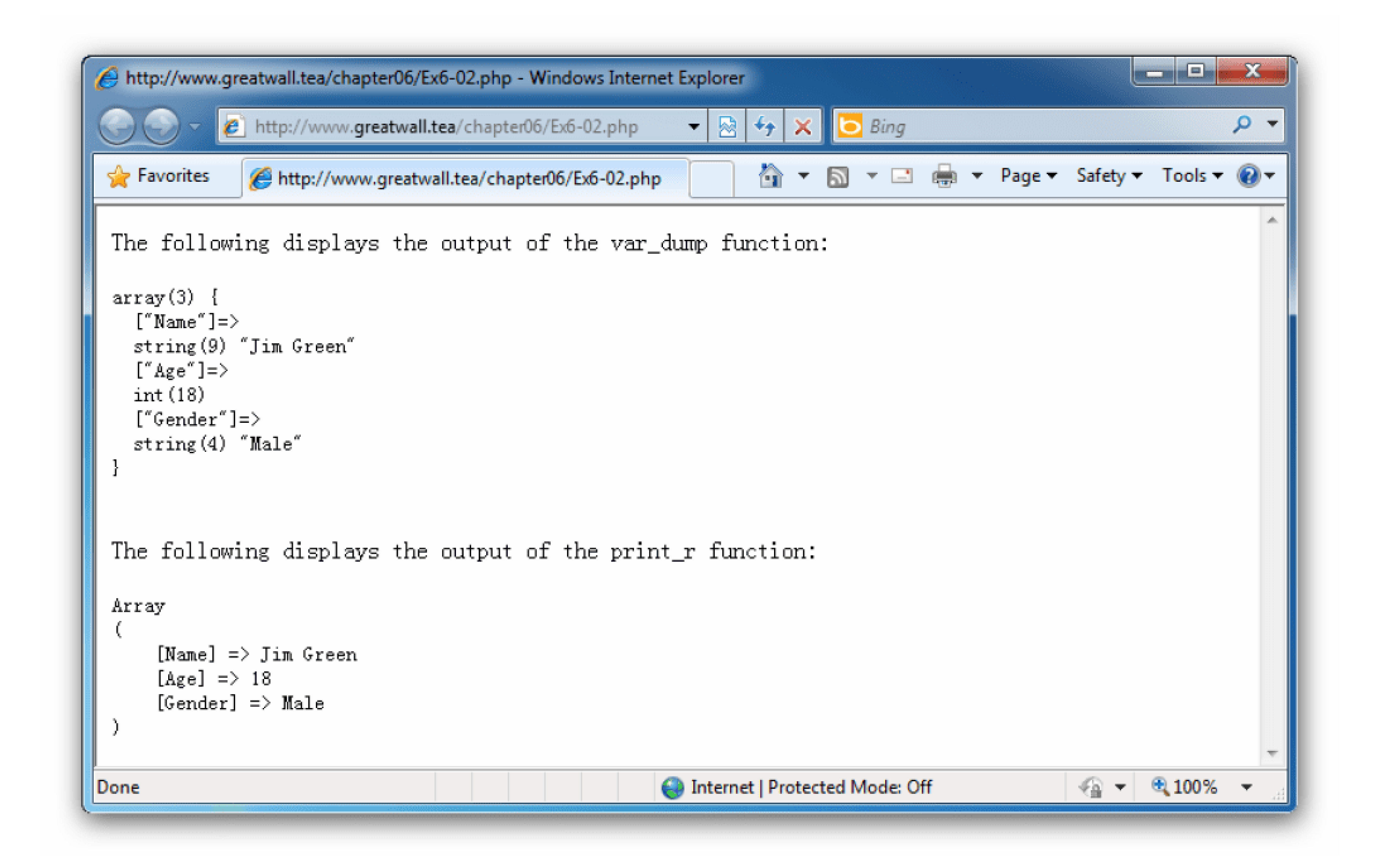


图 6-2　查看数据结构（二）

6.1.3　修改数组

数组一旦创建，其元素的值是可以随时修改的，就像可以随时修改某个变量的值一样。同样，也可以随时改变数组中元素个数，并复制整个数组。例如，有一个数组$capitals，可以使用以下的语句来定义这个数组：

```
$capitals = array("SD" => "Jinan",
"HEB" => "Baoding",
"HEN" => "Zhengzhou",
"HB" => "Wuhan",
"HN" => "Changsha",
"GD" => "Guangzhou",
"GX" => "Nanning");
```

在这个数组中，定义了中国若干省份的省会城市，其中河北省（HEB）的省会被误定义成了保定市。如果需要将其修改为正确的石家庄市，则可以使用下面这条语句：

```
$capitals["HEB"] = "Shijiazhuang";
```

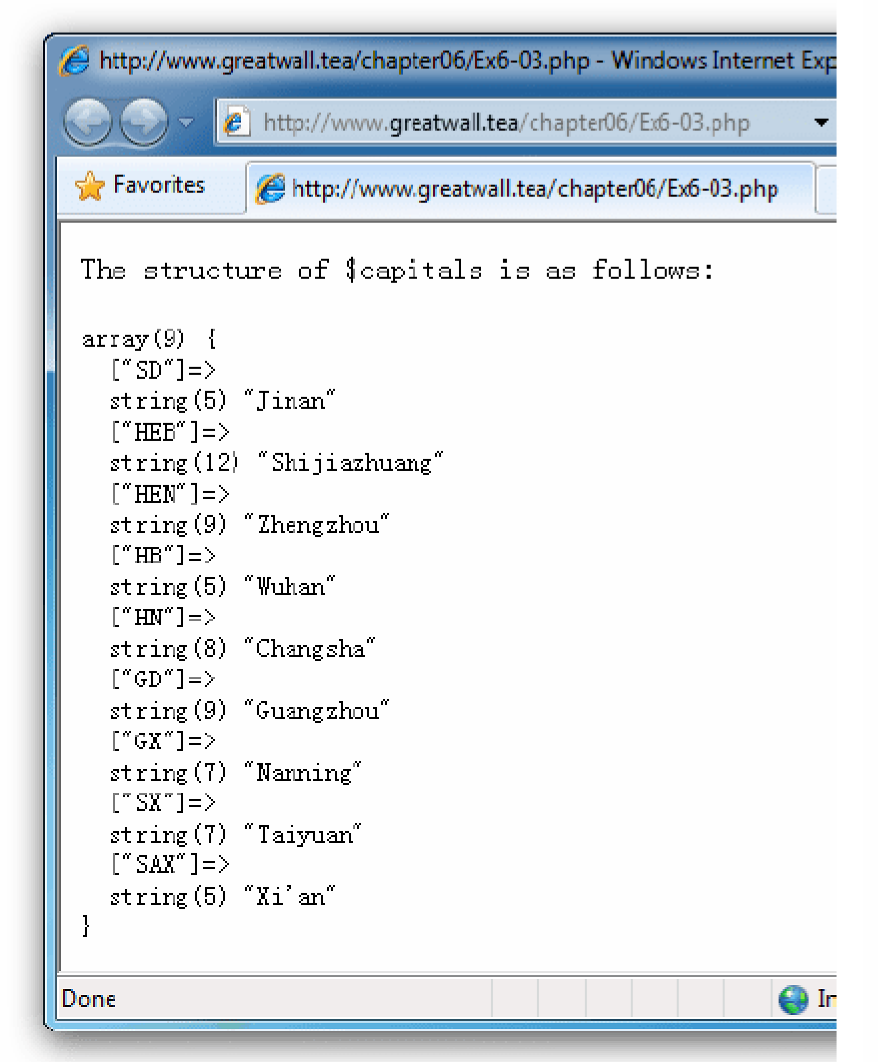
如果需要向数组$capitals 中添加山西省（SX）省会太原和陕西省（SAX）省会西安，则可以使用下面这两条语句：

```
$capitals["SX"] = "Taiyuan";
$capitals["SAX"] = "Xi'an";
```

如果需要为数组$capitals 做一个备份，即数组$capitals 现有的元素备份到数组$capitalsBackup 中，则可以使用下面这条语句：

```
$capitalsBackup = $capitals;
```

到此，我们一共创建了两个数组，一个为$capitals，另一个为$capitalsBackup。现在使用$var_dump 来查看一下这两个数组的结构，如图 6-3 所示。



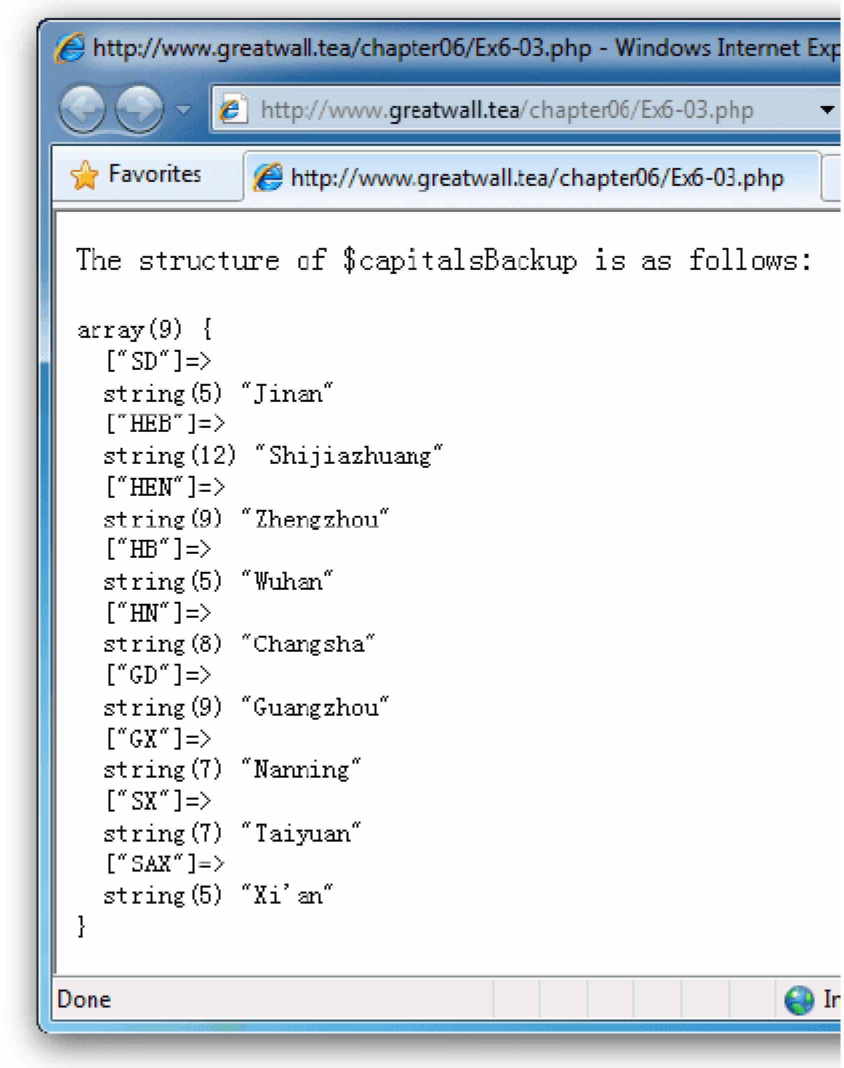


图 6-3　修改数组以及复制数组元素

如果需要移除数组$capitalsBackup 中的广东省（GD）的省会，可以使用下面的语句：

```
unset($capitalsBackup["GD"]);
```

当 PHP 引擎执行了这条语句之后，$capitalsBackup 这个数组中元素的个数就由原来的 9 个变成如图 6-4 所示的 8 个元素了。

我们可以再次使用 var_dump()函数检查一下，如图 6-4 所示。

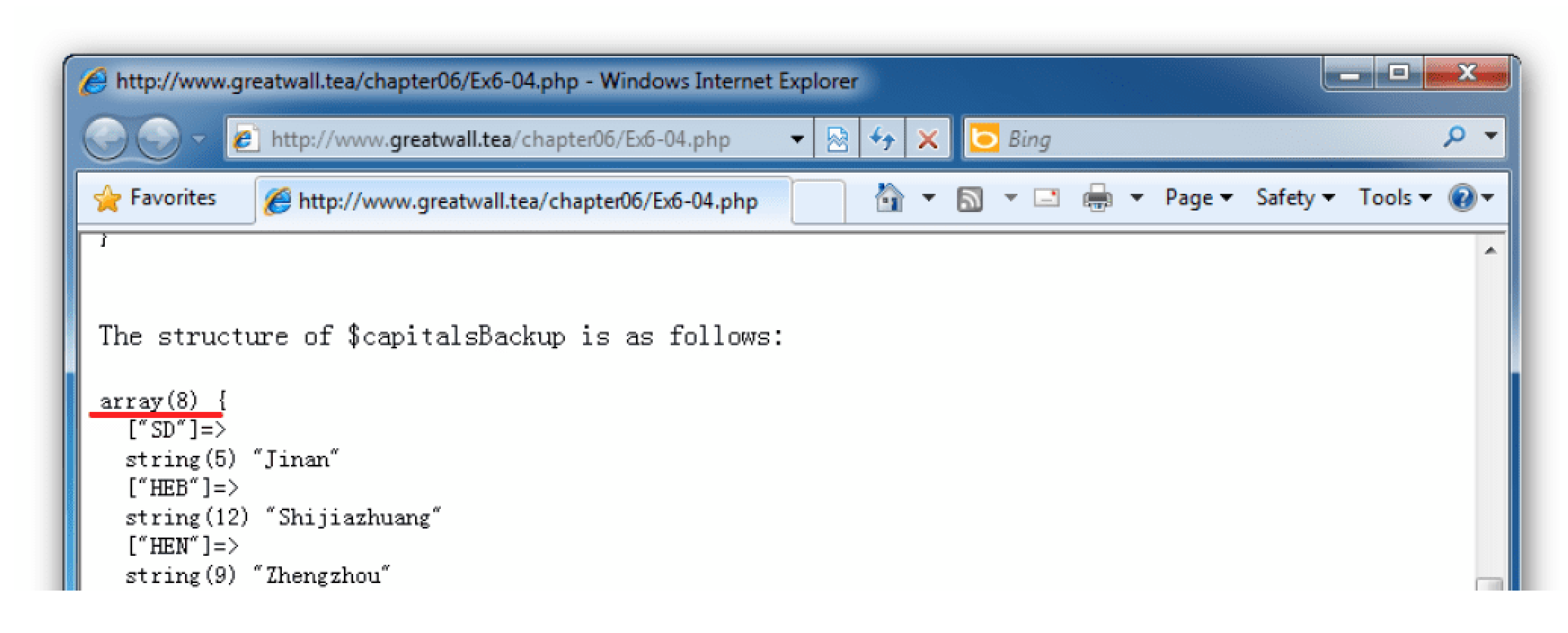


图 6-4　移除数组元素后

既然可以用 unset()函数来删除变量和数组元素，那是不是也可以用它来删除整个数组呢？答案是肯定的。比如，现在我们觉得$capitalsBackup 这个数组没有什么作用了，想删除它，那么可以执行下面的语句：

```
unset($capitalsBackup);
```

当再次使用 var_dump()函数显示数组结构时，只会得到如下所示的错误提示：

```
Notice:  Undefined variable: capitalsBackup in C:\greatwall\chapter06\
Ex6-05.php on line 31
```

提示信息指出 capitalsBackup 是个未定义变量。由此可知，$capitalsBackup 已经不存在了。有的同学可能还会有疑问，$capitalsBackup 不是个数组吗，为什么这里说它是个未定义变量呢？

对于这个问题，我们可以做如下解读：

$capitalsBackup 在未删除之前的确是个数组，但是删除之后，它都不存在了，PHP 处理引擎怎么会知道它是数组呢，自然也就把它当成变量了。

如果你对这个解答不太满意，那就看看本章的开头吧。在那里我曾说过这么一句话，那就是："其实，我们还真可以把数组看成是用于存放一组数据的复杂变量。"

数组其实就是一种特殊的变量。提示信息中称其为变量，理解起来还真没有什么障碍啊。

最后，我们把这一节里使用到的脚本整合一下，放在下面，供大家参考。

【例 6.3】 向数组中添加元素、修改数组元素和复制数组。

```
<?php
    $capitals = array("SD" => "Jinan",
                  "HEB" => "Baoding",
                  "HEN" => "Zhengzhou",
```

```
                     "HB" => "Wuhan",
                     "HN" => "Changsha",
                     "GD" => "Guangzhou",
                     "GX" => "Nanning");

    //把河北的省会改成石家庄
    $capitals["HEB"] = "Shijiazhuang";

    //添加太原和西安两个省会
    $capitals["SX"] = "Taiyuan";
    $capitals["SAX"] = "Xi'an";

    $capitalsBackup = $capitals;           //备份数组

    echo "The structure of \$capitals is as follows:<br>";
    echo "<pre>";
    var_dump($capitals);
    echo "</pre>";

    echo "<br>The structure of \$capitalsBackup is as follows:<br>";
    echo "<pre>";
    var_dump($capitalsBackup);
    echo "</pre>";
?>
```

【例 6.4】 删除数组中的指定元素。

```
<?php
    $capitals = array("SD" => "Jinan",
                     "HEB" => "Baoding",
                     "HEN" => "Zhengzhou",
                     "HB" => "Wuhan",
                     "HN" => "Changsha",
                     "GD" => "Guangzhou",
                     "GX" => "Nanning");

    $capitals["HEB"] = "Shijiazhuang";     //把河北的省会改成石家庄
    //添加太原和西安两个省会
    $capitals["SX"] = "Taiyuan";
    $capitals["SAX"] = "Xi'an";

    $capitalsBackup = $capitals;           //备份数组

    unset($capitalsBackup["GD"]);          //移除广东省的省会

    echo "The structure of \$capitals is as follows:<br>";
    echo "<pre>";
    var_dump($capitals);
    echo "</pre>";

    echo "<br>The structure of \$capitalsBackup is as follows:<br>";
    echo "<pre>";
    var_dump($capitalsBackup);
    echo "</pre>";
?>
```

【例 6.5】 删除整个数组。

```
<?php
    $capitals = array("SD" => "Jinan",
```

```
                    "HEB" => "Baoding",
                    "HEN" => "Zhengzhou",
                    "HB" => "Wuhan",
                    "HN" => "Changsha",
                    "GD" => "Guangzhou",
                    "GX" => "Nanning");

    $capitals["HEB"] = "Shijiazhuang";

    $capitals["SX"] = "Taiyuan";
    $capitals["SAX"] = "Xi'an";

    $capitalsBackup = $capitals;           //备份数组

    unset($capitalsBackup);                //删除数组

    echo "The structure of \$capitals is as follows:<br>";
    echo "<pre>";
    var dump($capitals);
    echo "</pre>";

    echo "<br>The structure of \$capitalsBackup is as follows:<br>";
    echo "<pre>";
    var_dump($capitalsBackup);
    echo "</pre>";
?>
```

6.2　排排坐——数组的遍历、排序与比较

通过上一节的学习，我们知道了如何创建数组、查看数组元素、修改数组元素、向数组中增加数组、删除数组元素以及删除数组，大体上了解了什么是数组。在本节里，我们将继续上一节的学习，了解如何遍历数组元素、给数组元素排序和比较任意两个数组。

6.2.1　如何遍历数组中的元素

“遍历”这个词看上去可能比较高级。所谓“遍历”，也就是依次对数组中的每个元素分别做些操作。例如，如果需要打印数组中的每一个元素或者把数组中的每一个元素分别存储进数据库中，这都需要用到数组的遍历。在本小节里，我们将学习两个遍历数组的方式：手动遍历和自动遍历。

1．手动遍历

所谓手动遍历，就是指使用脚本控制系统指针从指定数组的第一个元素开始，一个一个的向前移动。每移动一步，就做一些操作。这里，可以把数组看成是一张表，而每个数组元素就是一个表项，在表的左边有一个可以上下浮动的游标。我们可以用手滑动游标来指定不同的数组元素。

PHP 提供了很多简化手动遍历数组元素的函数，可以先来认识一下它们。

- current($arrayName)：指的是系统指针当前指向的数组元素，又称为当前指针。
- next($arrayName)：指的是当前指针指向元素的下一个数组元素。

- ❑ prev($arrayName)：指的是当前指针指向元素的上一个数组元素。
- ❑ end($arrayName)：指的是当前数组的最后一个数组元素。

在对一个数组做手动遍历操作时，除非你已经移动过系统指针，否则当前指针为该数组的第一个元素。如果之前已经移动过系统指针，而又想让当前指针为该数组的第一个元素，则可以在遍历数组之前使用 reset()函数，如：

```
reset($arrayName);
```

如果使用手动遍历的方法来遍历数组，我们需要用到一个赋值语句和一个输出语句来输出数组中的指定元素的值。现在，还是使用上一节中定义的数组$capitals 来举个例子。

【例 6.6】 手动遍历数组元素。

```
$capitals = array("SD" => "Jinan",
              "HEB" => "Baoding",
              "HEN" => "Zhengzhou",
              "HB" => "Wuhan",
              "HN" => "Changsha",
              "GD" => "Guangzhou",
              "GX" => "Nanning");

//输出当前指针指向的数组元素
$value = current($capitals);
echo "The current pointer is <b>$value</b>.<br>";

//输出当前指针指向的数组元素的下一个元素
$value = next($capitals);
echo "The next pointer is <b>$value</b>.<br>";

//输出数组的最后一个元素
$value = end($capitals);
echo "The last pointer is <b>$value</b>.<br>";

//输出当前指针指向数组元素的上两个元素
$value = prev($capitals);
$value = prev($capitals);
echo "The pointer two elements ahead of the current pointer is
<b>$value</b>.<br>";
```

这段脚本运行的结果如图 6-5 所示。

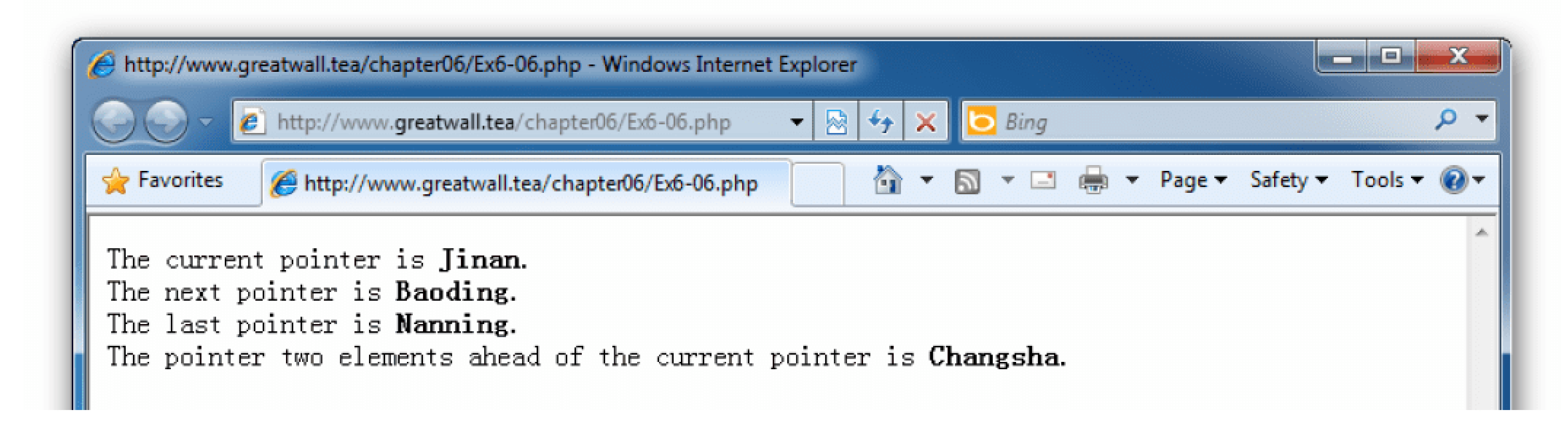


图 6-5　手动遍历数组元素

通过结果可知，第一组赋值和输出语句的结果显示当前系统指针指向数组$capitals 的第一个元素（即山东省的省会济南）。第二组赋值和输出语句的结果显示当前系统指针指

向当前指针的下一个元素（即河北省省会保定）。第三组赋值和输出语句的结果显示当前系统指针指向数组$capitals 的最后一个元素（即广西区首府南宁）。在第四组赋值和输出语句中，使用了两次 prev()函数，也就是连续两次向前移动了系统指针，通过图 6-5 所示中显示的结果可以看出，当前系统指针指向了数组$capitals 的倒数第三个元素（即湖南省省会长沙）。

由此可见，我们可以通过连续使用 next()、prev()和 end()函数来移动系统当前指针到数组中的任一位置，然后使用赋值和 echo 语句来输出当前指针。

虽然手动遍历数组元素看上去十分简单，但是对于动辄数万元素的数组来说却只能望洋兴叹了吧。没有关系，我们可以使用自动遍历数组元素的方法来对付庞大的数组。

2. 自动遍历

其实所谓的自动遍历就是使用在遍历数组时使用循环语句 foreach。使用该语句可以让系统一个接着一个的输出数组中的每个元素，一次输出一个。语句结构如下：

```
foreach ($arrayName as $key => $value)
{
    statements
}
```

其中，

- arrayName 指的是需要遍历的数组名称，在例 6.7 中，我们将继续使用 6.1.3 小节中定义的数组$capitals。
- key 指的是用于存储数组索引的变量名。这个参数是可选的。在例 6.7 中，我们将使用变量$province 来存储数组索引。
- value 指的是用于存储数组元素值的变量名。在例 6.7 中，我们将使用$capital 来存储数组元素的值。

下面，我们来看一段脚本，学习一下如何自动遍历指定数组。

【例 6.7】 自动遍历数组元素。

```
$capitals = array("SD" => "Jinan",
              "HEB" => "Shijiazhuang",
              "HEN" => "Zhengzhou",
              "HB" => "Wuhan",
              "HN" => "Changsha",
              "GD" => "Guangzhou",
              "GX" => "Nanning");

//使用 foreach 语句来遍历数组
foreach ($capitals as $province => $capital) {
    echo "The capital of $province is $capital.<br>";

}
```

这段脚本的运行结果如图 6-6 所示。

通过如图 6-6 所示的结果，可以知道，变量$province 存储了系统当前指针指向元素的索引，而$capital 则存储了系统当前指针指向元素的值。通过 foreach 语句一条一条地将数组中的元素按照指定的格式打印了出来。关于 foreach 语句的具体内容可以参见第 8 章的条件与循环。

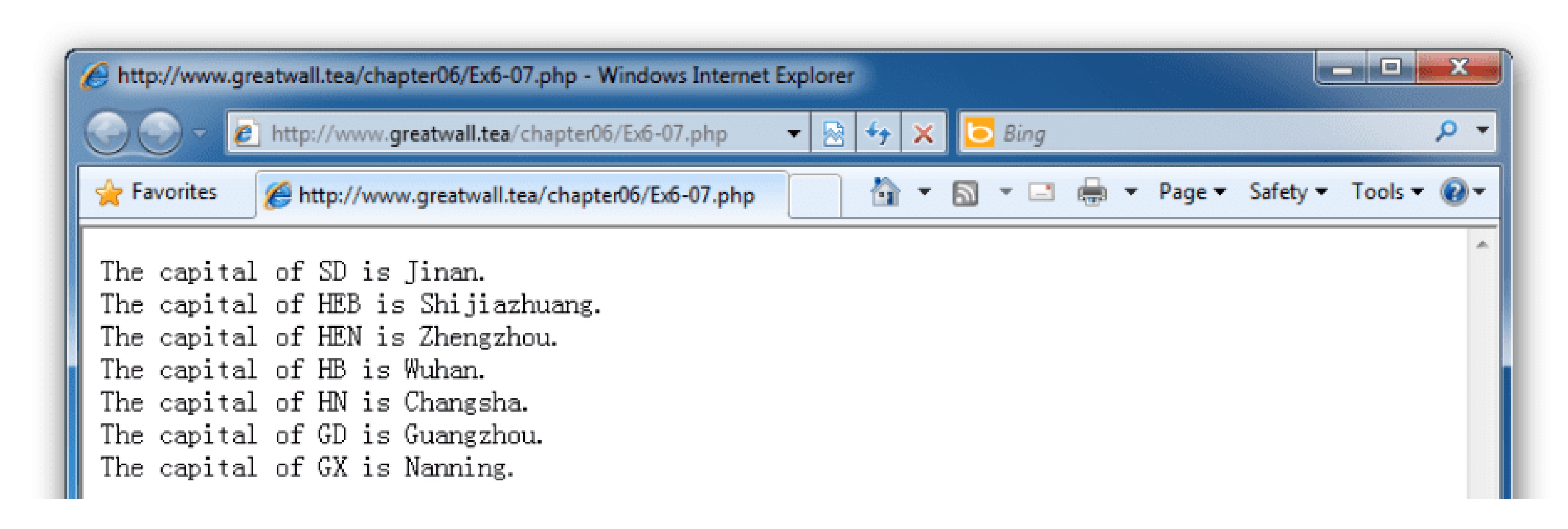


图 6-6　自动遍历数组元素

通过比较手动和自动遍历数组的方法可知，手动遍历适合于小规模的数组遍历，可以引导系统指针灵活地在数组元素间来回移动；而自动遍历适合于大规模的数组遍历，可以引导系统指针按照一定顺序逐一地在数组元素间移动。

6.2.2　如何给数组中的元素排序

在对某数组排序之前，PHP 是按照各数组元素被创建的先后顺序来排序的。也就是说，如果不改变数组中各元素的顺序而直接输出数组中的元素的值，那么 PHP 处理引擎将按照各元素被创建的先后顺序来输出各元素的索引和对应的值。这一点已经在例 6.7 和图 6-6 所示中得到了印证。

在编写 PHP 脚本时，可以按索引或元素值对数组中的元素进行排序。为此，PHP 还提供了一些用于数组元素排序的函数。下面就来认识一下它们。

1. sort($arrayName)

该函数将按照指定数组中各元素的值的大小对数组元素进行排序。若元素值同为数字，则按照数字大小排序，若元素值中既有数字，也有字符串，则数字排在字符串前，字符串按照大小写和字母顺序排序。同一字母的大写字母排在小写字母前面。需要注意的是，若数组的索引为字符串，如例 6.7 所示中定义的数组$capitals，在使用 sort()函数后，字符串索引将全部转换成数字。

在图 6-7 所示中，可以看到，在使用了 sort()函数对数组$capitals 中的元素排序之后，第一个创建的数组元素 Jinan 被排到了第 3 位，而第五个创建的元素 Changsha 则被排到了第 1 位。Zhengzhou 则因为首字母是 Z 而被排到了数组的最后一位。同时，数组元素的索引也由之前字符串变成了图 6-7 所示中的数字。

如果想保留数组的字符串形式的索引，则需要使用 asort()方法。

2. asort（$arrayName）

该函数与 sort()函数一样，将按照指定数组中各元素值的大小对数组元素进行排序，数字在前，字符串在后。数字按照数字大小排序，字符串按照字母顺序和大小写排序。它与 sort()函数唯一不同的就是，在使用 asort()函数对指定数组排序后，数组各元素的原索引会予以保留。也就是说图 6-7 中被数字取代的数组元素索引会变成如图 6-8 所示的样子：数组$capitals 中的各元素按照元素值的大小排序，但是数组元素的原索引却不再被数字取代，

而是得到了保留。

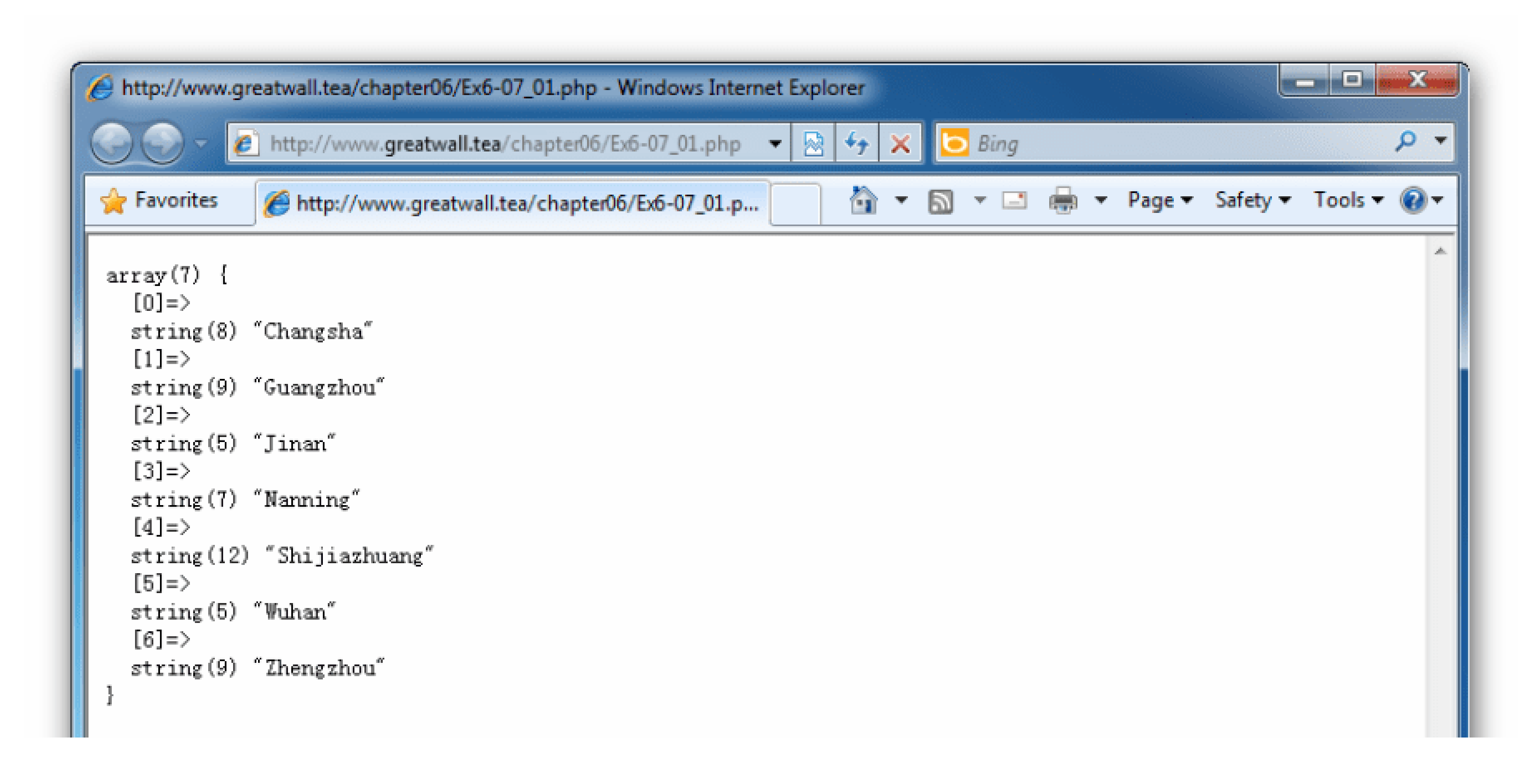


图 6-7　使用 sort()函数的效果

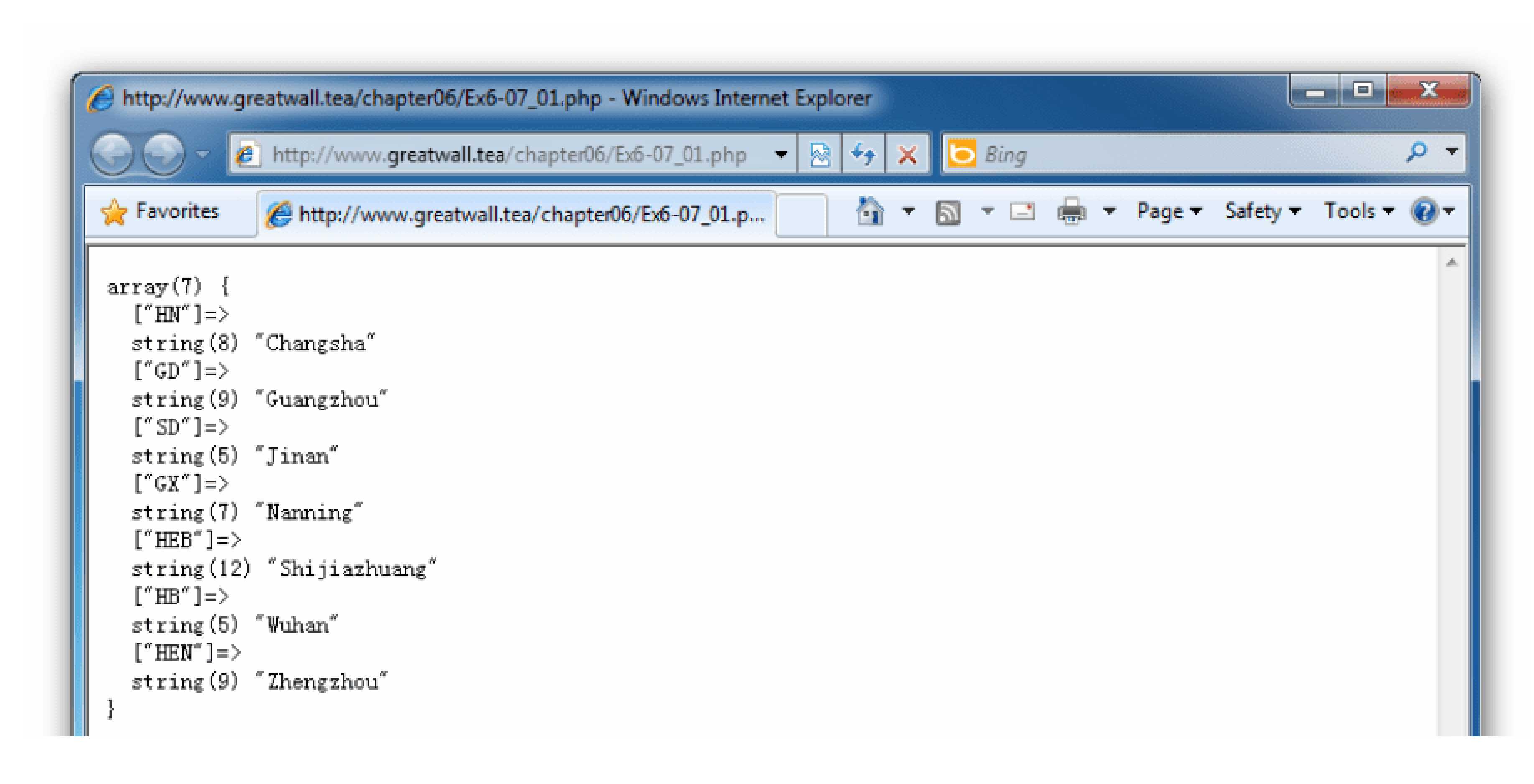


图 6-8　使用 asort()函数的效果

与 sort()和 asort()函数类似，PHP 还提供了一对按照指定数组中各元素值从大到小进行排序的函数：rsort()和 arsort()。其中 r 代表 reverse，也就是反向的意思。我们若使用 rsort()和 arsort()函数对数组$capitals 中的元素进行排序，得到的结果将如图 6-9 所示。在图 6-9 中，左边为使用 rsort()函数的效果，而右边则为使用 arsort()函数的效果。

据结果显示，rsort()将指定数组中各元素的值按照首字母在字母表中的先后顺序从后往前进行了排序，于是之前在图 6-7 中排末位的 Zhengzhou 排在了第 1 的位置，而之前在图 6-7 中排首位的 Changsha 则排到了末位。与图 6-7 所示类似的是，数组各元素的索引按照排序的结果变成了从 0 到 6 的数字了。而 arsort()函数与 rsort()函数一样，也将各元素的值按照首字母在字母表中的先后顺序从后往前进行了排序。但是，与 rsort()函数不同的是，arsort()函数保留了指定数组各元素原有的字符串形式的索引。

除此之外，PHP 还提供了另外一对函数：ksort()和 krsort()。其中，k 代表 key，也就是说这一对函数是按照数组元素的索引值对数组元素进行排序的。图 6-10 所示展示了使用这两个函数的效果。

在图 6-10 所示中，左边为使用 ksort()函数的效果，而右边为使用 krsort()函数的效果。

在使用 ksort()函数对数组$capitals 进行排序之后，排在第一位的为 Guangzhou，因为广东省（GD）的首字母是该数组中所有省份名称的首字母在字母表中最靠前的一个；而 Jinan 排在了最后，则是因为山东省（SD）的首字母是该数组中所有省份名称的首字母在字母表中最靠后的一个。

若使用 krsort()函数对数组$capitals 中的元素进行排序，得到的结果则与使用 ksort()相反。

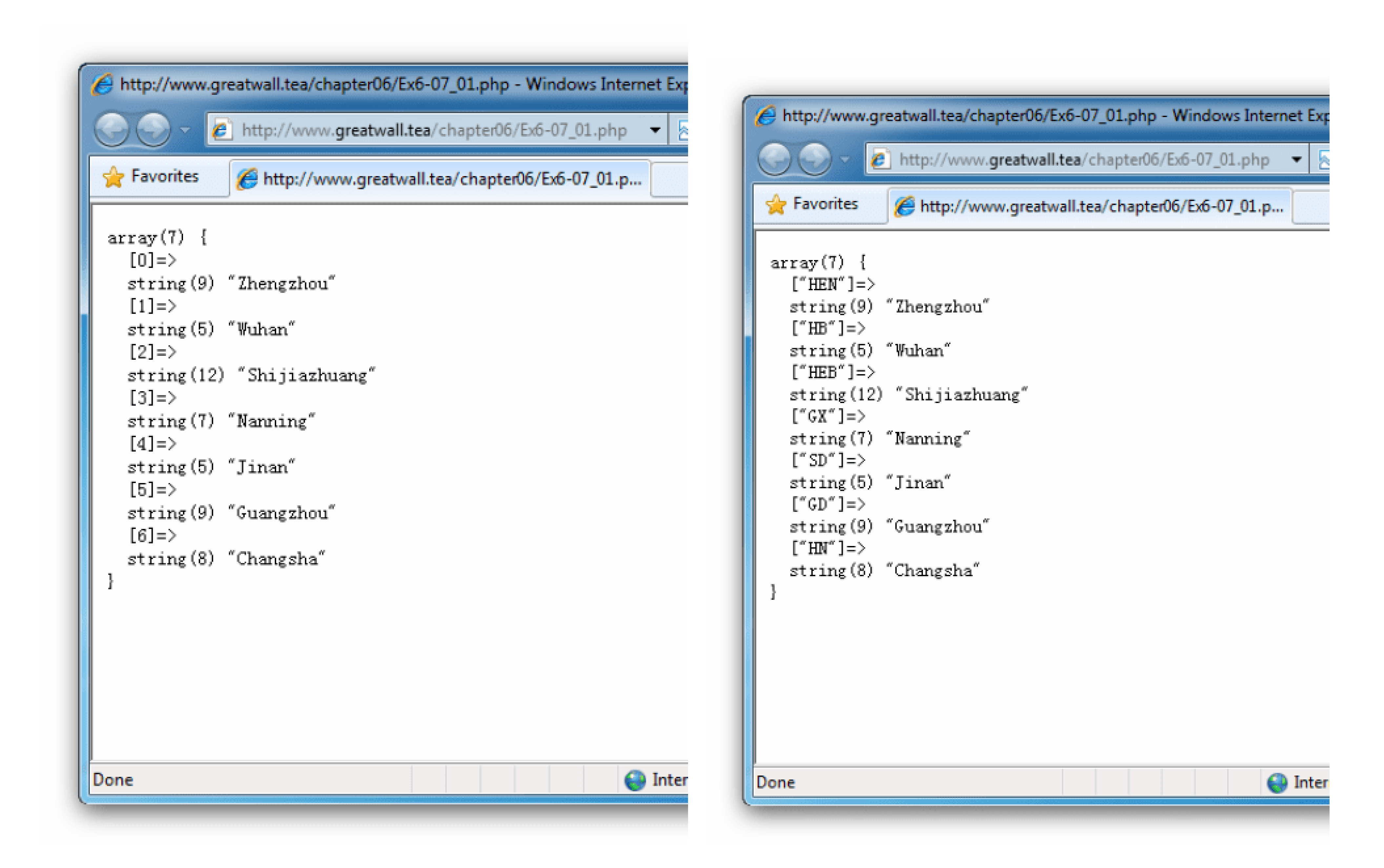


图 6-9　使用 rsort()和 arsort()函数的效果

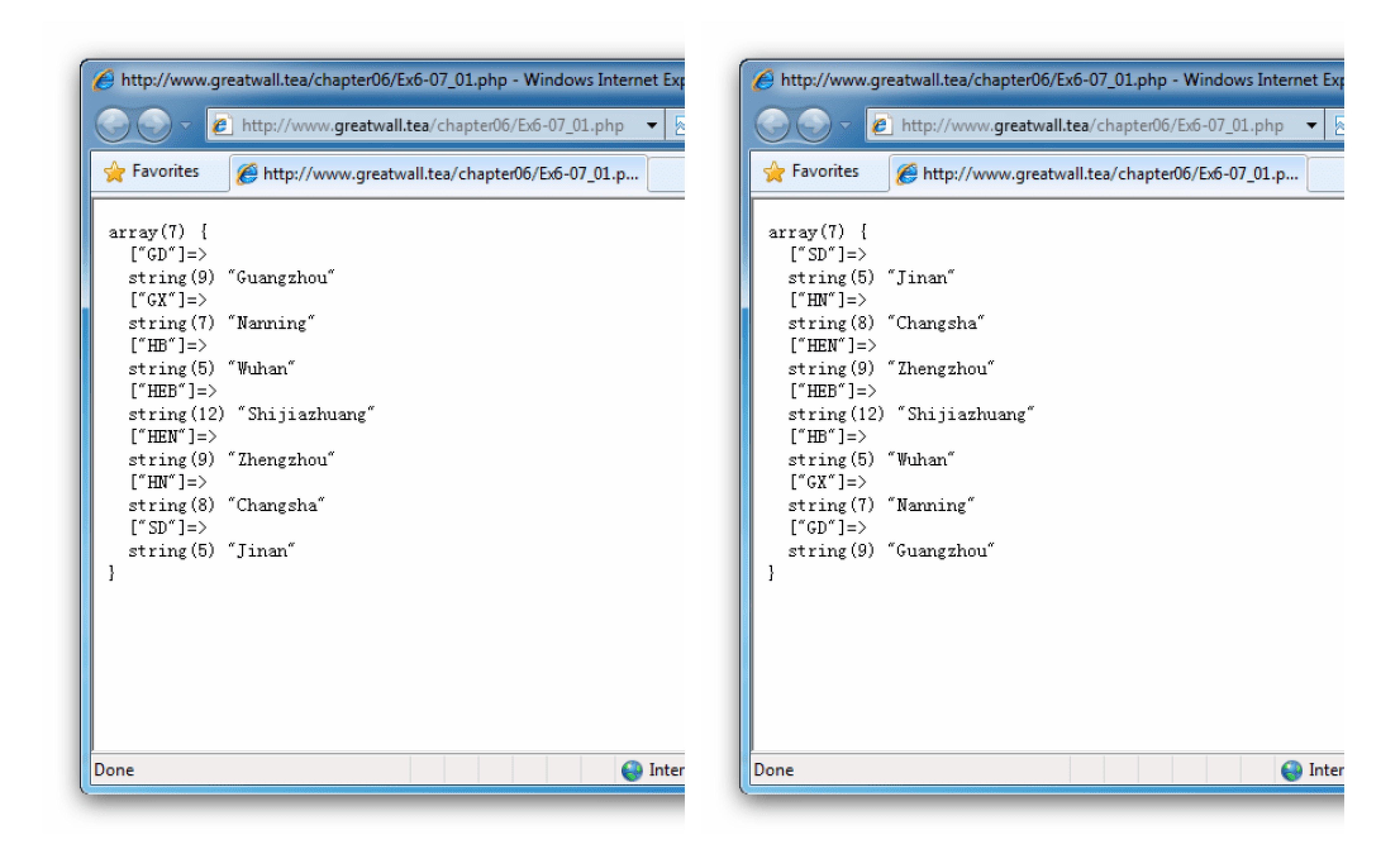


图 6-10　使用 ksort()和 krsort()函数的效果

表 6-1 对这三组六个函数进行一下总结。

表 6-1　数组排序函数

排 序 函 数	排序效果说明
sort($arrayName)	按元素值的大小正向排序，并按排序结果替换元素的索引
asort($arrayName)	按元素值的大小正向排序，并保留元素原始索引
rsort($arrayName)	按元素值的大小反向排序，并按排序结果替换元素的索引
arsort($arrayName)	按元素值的大小反向排序，并保留元素原始索引
ksort($arrayName)	按元素索引的大小反向排序
krsort($arrayName)	按元素索引的大小反向排序

在这一节中，使用到的脚本如下，供大家参考。

【例 6.8】 使用数组排序函数为数组元素排序。

```
1   <?php
2       $capitals = array("SD" => "Jinan",
3                    "HEB" => "Shijiazhuang",
4                    "HEN" => "Zhengzhou",
5                    "HB" => "Wuhan",
6                    "HN" => "Changsha",
7                    "GD" => "Guangzhou",
8                    "GX" => "Nanning");
9
10      sort($capitals);    //按元素值的大小正向排序，并按排序结果替换元素的索引
11      #asort($capitals);  //按元素值的大小正向排序，并保留元素原始索引
12      #rsort($capitals);  //按元素值的大小反向排序，并按排序结果替换元素的索引
13      #arsort($capitals); //按元素值的大小反向排序，并保留元素原始索引
14      #ksort($capitals);  //按元素索引的大小反向排序
15      #krsort($capitals); //按元素索引的大小反向排序
16      echo "<pre>";
17      var_dump($capitals);
18      echo "</pre>";
19  ?>
```

大家可以移除第 10 行～第 15 行相应函数前的注释符号（#）来激活该函数，或在相应函数前添加注释符号（#）使该函数失效。务必保证一次只激活一个函数。若多条函数被同时激活，则最后激活的函数生效。

6.2.3　如何比较数组

有时候，我们可能需要比较两个数组中的元素是否相同。这时可以使用 array_diff()函数，这个函数可以带两个或两个以上的参数。还可以把使用 array_diff()函数得到的两个或多个数组的比较结果直接赋值给一个新的数组。

【例 6.9】 使用 array_diff()函数比较两个数组中的元素是否相同。

```
$array1 = array("a" => "apple", "b" => "banana", "c" => orange );
$array2 = array("grapefruit", "banana", "orange");
$diffArray = array_diff(array1, array2);
echo "<pre>";
var_dump($diffArray);
echo "</pre>";
```

在如例 6.9 所示的脚本中，定义了两个数组：array1 和 array2。其中，array1 中使用了字符串做元素的索引，而 array2 中的每个元素则会被 PHP 处理引擎自动分配一个数字索引。接着使用了 array_diff()函数比较这两个数组的不同，并将不同的元素存入 diffArray 数组。最后使用 var_dump()函数输出数组 diffArray 的结构。

在运行这个脚本之后中，可以发现，数组 diffArray 只有一个元素：apple。它的索引为 a，因为这个元素只存在于数组 array1 中。如果把例 6.9 中的第 3 行换成下面的样子：

```
$diffArray = array_diff(array2, array1);
```

那么数组 diffArray 中也只有一个元素。不过，这次不是 apple，而是 grapefruit 了。它的索引也不是 a，而是 0 了。

从这个示例中，可以知道，array_diff()函数的作用是找出第一个数组中与第二个数组不一样的元素，比较结果中的各元素的索引保持不变。需要注意的是，array_diff()函数会将不同数组中值相同的元素看成是相同的元素，无所谓这些元素的索引是否相同，就像例 6.9 中的 banana 和 orange 一样。

如果想找出两个数组中元素索引或元素值不一样的所有元素，则可以使用 array_diff_assoc()函数。其中 assoc 是英文单词 associate 的缩写，表示关联性索引，也就是之前提到的字符串索引。

现在，把例 6.9 中的第 3 行换成下面的样子，看看运行的结果有什么不同：

```
$diffArray = array_diff_assoc(array1, array2);
```

在运行了修改好的脚本后，数组 diffArray 中的元素变成了三个，它们分别是索引为 a 的 apple、索引为 b 的 banana 和索引为 c 的 orange。因为数组 array1 和 array2 中的三个元素要么索引不一样，要么元素值不一样，要么索引和值都不一样。

PHP 除了提供这些用于找不同函数之外，还提供了另外一对函数：array_intersect()和 array_intersect_assoc()，其中 intersect 的意思是交叉。这两个函数的作用是找出两个或多个数组的交集，前者只要求元素值相同即可，而后者则要求元素索引和元素值都相同才行。

现在把例 6.9 中的第 3 行换成如下的样子：

```
$simArray= array_intersect(array1, array2);
```

然后把第 5 行改成如下的样子：

```
var_dump($simArray);
```

这时候，数组 simArray 中的元素个数应该为两个。它们分别是索引为 b 的 banana 和索引为 c 的 orange。

如果把例 6.9 中的第 3 行换成如下的样子：

```
$simArray= array_interset_assoc(array1, array2);
```

那么，simArray 将不含任何元素，因为这两个数组中的每个元素要么索引不同，要么值不同，要么索引和值都不相同。

最后，我们还像上一节一样对这两组四个函数进行一下总结，如表 6-2 所示。

表 6-2　数组比较函数

比 较 函 数	效 果 说 明
array_diff($array1, $array2, …)	找出第一个数组中与第二个数组元素值不同的元素
array_diff_assoc($array1, $array2, …)	找出第一个数组中与第二个数组元素值或索引不相同的元素
array_intersect($array1, $array2, …)	找出第一个数组中与第二个数组元素值相同的元素
array_intersect_assoc($array1, $array2, …)	找出第一个数组中与第二个数组元素值和索引均相同的元素

6.3　串串门——数组与其他数据类型的互转

说句实话，人是个奇怪的动物：一旦可以完成一件事，就会想着怎么把这件事情做出花样来。如果我告诉你数组可以被转换成其他的数据类型。虽然可能你不懂如何转换，但你肯定会问我：那是不是意味着其他的数据类型也可以被转换成数组？

不管你问了没问，这都不重要了。在本节里，我们就来讨论一下为什么要翻来倒去地折腾着把数组变成这又变成那，又把不是数组的变量转换成数组，以及如何实现这种转换。

6.3.1　为什么要转换

数组，我们所说的复杂变量，它的组织结构说紧密也谈不上紧密，因为想要往里添加些元素或者从中间删除些元素看上去是那么的简单；但是说松散也不见得有多松散，因为想把它们拆开，一个一个地处理也不是一件那么简单的事情。说白了，数组存储的就是一串信息元素。

有时候，我们需要把数组转换成一个字符串，而这个字符串包含该数组的所有的元素的值；有时候，我们需要把有着特定规律（分隔符）的一串字符划开，把这串字符中的某些内容当成数组的元素，一个一个地存储起来。只要确定了有分隔符，这些转换对于 PHP 来说就是易如反掌。

其实，数组不光可以被转换成字符串，也不是只有字符串才可以被转换成数组。我们可以把任意的一个数组拆开变成一个个松散独立的变量，这时数组中的元素索引和元素值分别变成了独立的变量名和变量值；我们也可以把若干的变量放在一起组成一个数组，这独立的变量名和变量值也就变成了这个数组中各元素的元素索引和元素值。

那么为什么要这么干呢？

我们转换来、转换去的理由说千道万，总结起来只有一句话，那就是“形势需要”。例如，可以将用户的姓名、性别、年龄和籍贯之类的基本信息存储在一个数组中，但是当需要在不同的页面间传递这些数据的时候，可以选择将这些信息整合成一个字符串，也可以选择将这些信息按照项目的不同转换成若干个字符串，用于信息的加密。再比如，在某个页面接收到了从其他页面传递过来的经过转换和加密了的信息，这时，就需要对接收到的信息进行解密和重组，以便还原成可以处理的数据。

可能你对于上述的情景还会有其他的解决办法，但已经无关乎是否需要转换来、转换去这件事本身了。在下面两小节的内容中，我们将学习：

- 如何拆分字符串成数组，又如何整合数组元素成字符串；

- ❑ 如何拆分数组成变量，又如何整合变量成数组。

6.3.2　数组与字符串的互转

在本小节里，我们用数组$capitals 来学习如何整合数组元素成字符串，以及如何拆分字符串成数组。

【例 6.10】 拆分字符串成数组。

```
<?php
    $capitals = array("SD" => "Jinan",
                  "HEB" => "Shijiazhuang",
                  "HEN" => "Zhengzhou",
                  "HB" => "Wuhan",
                  "HN" => "Changsha",
                  "GD" => "Guangzhou",
                  "GX" => "Nanning");

    $stringIn = implode ("|", $capitals);
    echo $stringIn;
?>
```

在这段脚本中，在定义了数组$capitals 之后，又定义了一个变量$stringIn。然后使用 implode()函数为变量$stringIn 赋值，最后输出变量$stringIn 的值。其中 implode()函数可以将数组中各元素的值整合在一个字符串中。

这段脚本输出的结果如下：

```
Jinan|Shijiazhuang|Zhengzhou|Wuhan|Changsha|Guangzhou|Nanning
```

通过观察脚本运行的结果可以发现，数组$capitals 中各元素的值被“|”隔开了，而“|”就是 implode()函数中定义的分隔符。在 implode()函数中，可以使用任意字符串做分隔符。

接下来，来看看如何把变量$stringIn 还原成数组。

【例 6.11】 拆分字符串成数组。

```
$stringIn = " Jinan|Shijiazhuang|Zhengzhou|Wuhan|Changsha|Guangzhou|Nanning";
$arrayOut = explode("|", $stringIn);

echo "<pre>";
var_dump($arrayOut);
echo "</pre>";
```

在这段脚本中，定义了一个新的数组$arrayOut，并用 explode()函数为其赋值。在使用 explode()函数时，也定义了一个分隔符。

这段脚本的输出的内容如图 6-11 所示。

通过截图可以看到，变量$stringIn 的值被拆分重组成了一个拥有七个元素的数组，这些元素的值与$capitals 中各元素的值相同，但是索引却不一样。原来使用 explode()函数生成的数组，其元素索引是由系统按照各元素被添加进数组的先后顺序自动分配的。这一点请大家务必要注意。

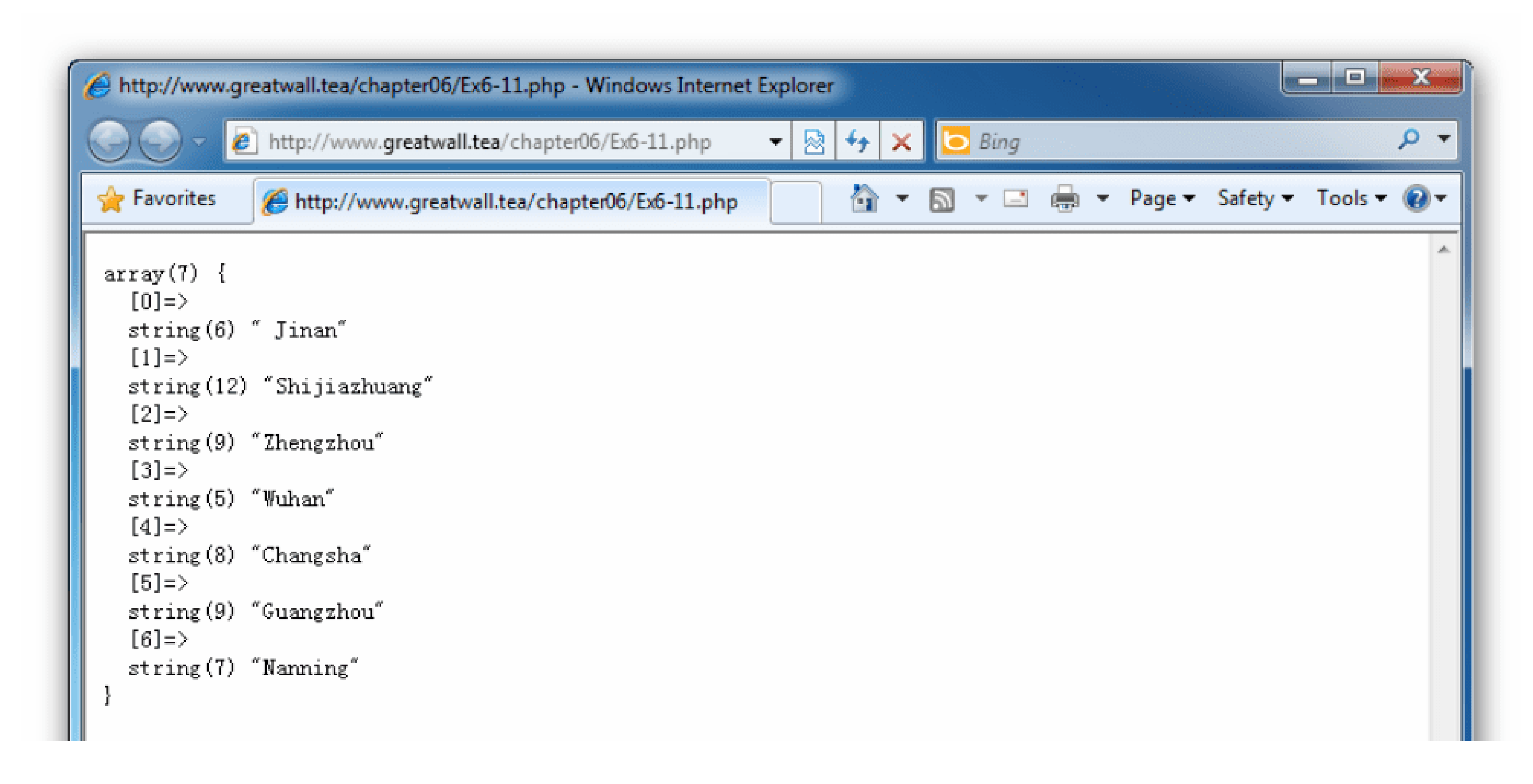


图 6-11　拆分字符串成数组

6.3.3　数组与变量的互转

除了将指定数组中的所有的元素整合到一个变量中，我们还可以将指定数组中的每个元素变成一个独立的变量。这样一来，各元素的原始索引和元素值都得到了保留。

在本小节里，还是继续使用数组$capitals 来举例。

【例 6.12】 将数组拆分成若干个独立的变量。

```
<?php
    $capitals = array("SD" => "Jinan",
                  "HEB" => "Shijiazhuang",
                  "HEN" => "Zhengzhou",
                  "HB" => "Wuhan",
                  "HN" => "Changsha",
                  "GD" => "Guangzhou",
                  "GX" => "Nanning");

    extract ($capitals);
    echo "The capital of Shandong is $SD.<br>";
    echo "The capital of Henan is $HEN.";
```

在这段脚本中，定义了含有七个元素的数组$capitals。然后使用了 extract()函数将数组$capitals 拆分成了七个独立的变量。这些变量的变量名为数组元素的原始索引，而变量值则为这些原始索引对应的元素值。接下来使用 echo 命令输出了两句话，在这两句话中引用了两个被拆分出来的变量名。

这段脚本运行的结果如下：

```
The capital of Shandong is Jinan.
The capital of Henan is Zhengzhou.
```

我们可以在这段脚本的末尾加上如下的内容来把这些个被拆分出来的变量再整合起来：

```
    $capitalsReunion = compact("SD","HEB","HEN","HB","HN","GD","GX");
    echo "<pre>";
```

```
15      var dump($capitalsReunion);
16      echo "</pre>";
17  ?>
```

在脚本的第 13 行，定义了一个新的数组$capitalsReunion，然后使用 compact()函数为其赋值。大家注意，在 compact()函数中，变量名是以字符串形式书写的，前面没有“$”号。如果大家觉得在 compact()函数中写这么一长串很麻烦的话，可以在前面再加上下面这一句：

```
$arrayIn = array("SD","HEB","HEN","HB","HN","GD","GX");
```

然后把第 13 行改成：

```
$capitalsReunion = compact($arrayIn);
```

这里，还是要提醒大家注意一下，compact()函数中引用的数组必须使用“$+数组名”这样的形式，而引用的变量则必须使用字符串的形式，不要在前面添加“$”号。

现在，还像之前各小节一样，总结一下在这一小节里学到的两组四个函数，如表 6-3 所示：

表 6-3　数组转换函数

比 较 函 数	效 果 说 明
implode(*delimiter*, $*array*)	使用指定的分隔符将指定数组中各元素的值组合成一个字符串
explode(*delimiter*, $*string*)	在指定的字符串中两个分隔符之间的内容做为一个元素添加到新建的数组中
extract($*array*)	将指定数组的各元素拆分成独立的变量，其中元素索引为变量名，元素值为变量值
compact(varname, $arrayName, …)	将指定变量组合成一个新数组，使用这些变量的变量名为新数组各元素的元素索引，这些变量的值为新数组各元素的值

最后看看这段脚本输出的结果，如图 6-12 所示。

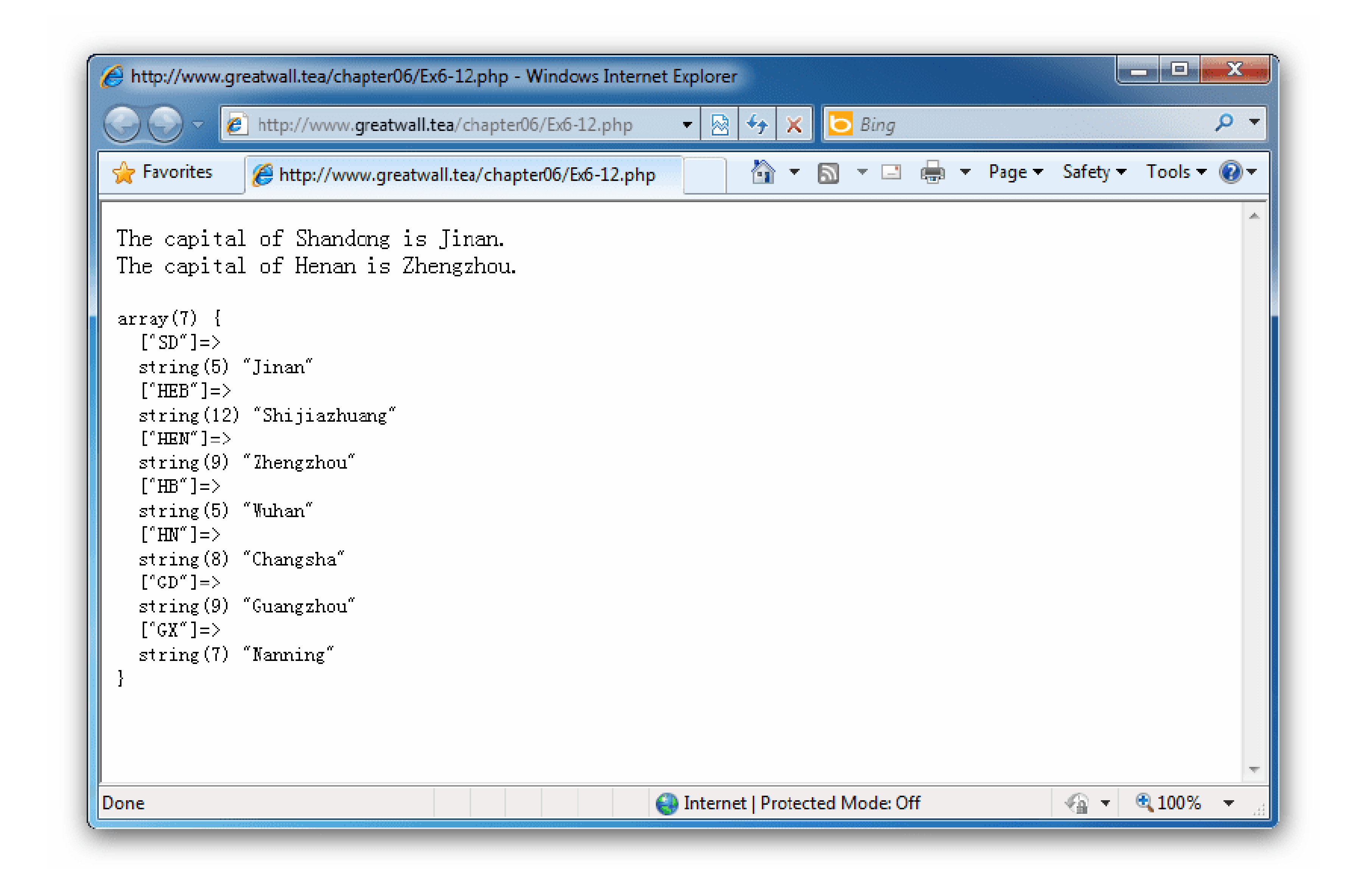


图 6-12　数组与变量的互换

6.4　分分合合——数组的拆分与合并

天下熙熙，皆为利来，天下攘攘，皆为利往。分分合合实乃最常见的需求了，有分就有合，有合必有分。在上一节里，我们学习了如何把数组转换成一个或多个变量以及如何把一个变量或若干个变量整合成数组。在本节里，我们将学习更加灵活的数组变换方式：数组的拆分和合并。

6.4.1　如何拆分数组

不知大家是否还记得在第 5 章的第 5.3.1 小节的文字游戏中学习过一招叫“断章取义”。说的是用 substr()函数截取一个字符串中的某一部分生成一个新的字符串。它的格式如下：

```
substr($stringName,start_pos,substr_len)
```

其中，$stringName 代表一个指定的字符串变量名，start_pos 代表子字符串在母子字符串中的起始位置，而 substr_len 代表子字符串的长度。

拆分数组使用到的函数与 substr()函数类似，这个函数叫 array_slice()，slice 就是切片的意思。它的格式如下：

```
array_slice($arrayName, offset, subArray_len, preserve_keys);
```

其中，$arrayName 代表一个指定的数组名；offset 指的是子数组首个元素相对于母数组首个元素的偏离值，也就是子数组的起始位置，subArray_len 指的是子数组包含的元素个数，而 preserve_keys 是个 Boolean 类型的参数，如果需要为子数组各元素中保留其在母数组中的原始索引，可将 preserve_keys 设为“TRUE”，否则将其设为“FALSE”。

需要注意的是，preserve_keys 这个变量的取值默认为“TRUE”，也就是说在没有特殊需要的时候，可忽略这个变量的设置。

在本小节里，还是以数组$capitals 为例来看看如何从这个数组中拆分出一个子数组。

【例 6.13】 拆分数组。

```
<?php
    $capitals = array("SD" => "Jinan",
                  "HEB" => "Shijiazhuang",
                  "HEN" => "Zhengzhou",
                  "HB" => "Wuhan",
                  "HN" => "Changsha",
                  "GD" => "Guangzhou",
                  "GX" => "Nanning");

    $subArray = array_slice($capitals, 1, 5);
    echo "<pre>";
    var_dump($subArray);
    echo "</pre>";
?>
```

在这一段脚本中，定义了一个拥有七个元素的数组$capitals 以及一个新数组$subArray，

然后使用 array_slice()函数截取了从数组$capitals 第二位开始一共 5 个元素并将它们加入到数组$subArray 中。最后使用 var_dump()函数输出了这个子数组的结构，结果如图 6-13 所示。

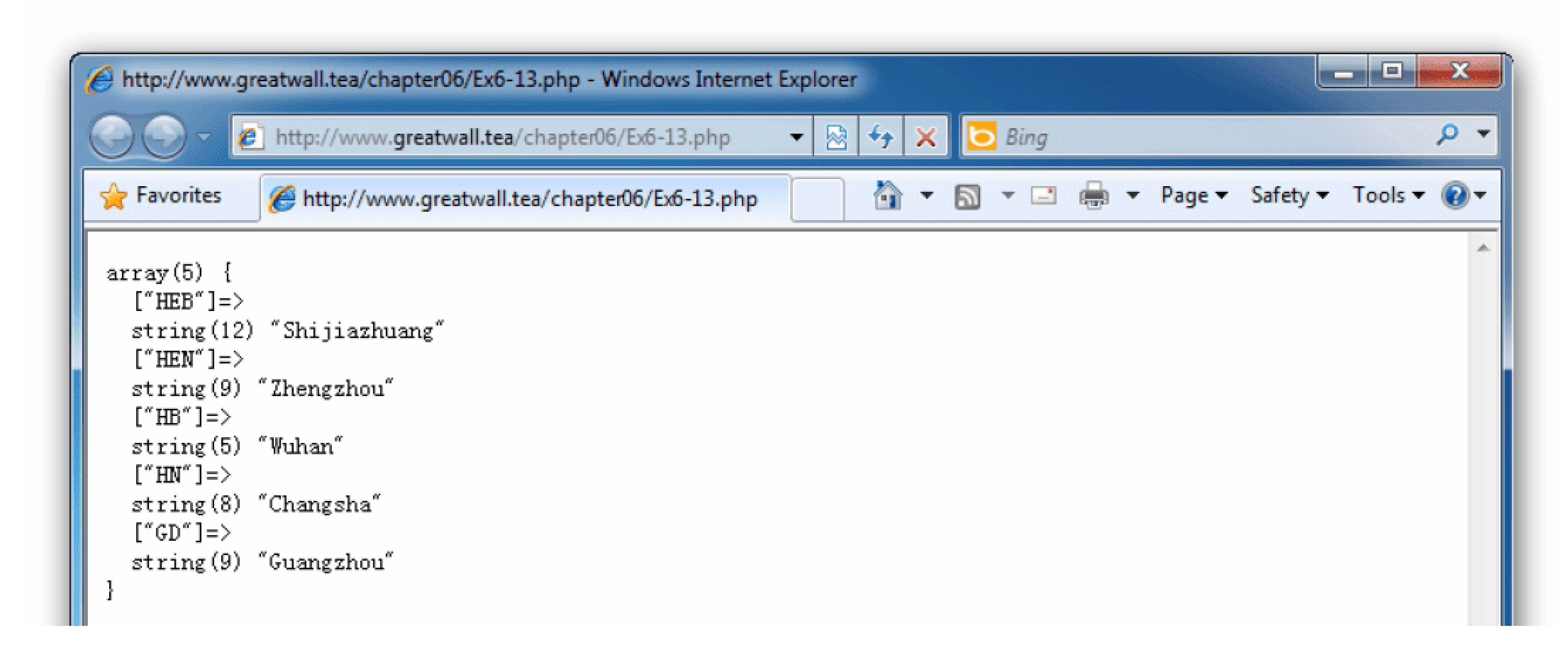


图 6-13　拆分数组

在图 6-13 所示中，我们可以发现子数组$subArray 一共拥有五个元素，第一个元素为母数组的第二个元素，最后一个元素为母数组的第六个元素。而母数组中的第一个元素和最后一个元素都被截掉了。

6.4.2　如何合并数组

就像在本节开头说的那样：天下之事，有合就有分，有分必有合。那么在这一小节里，就看看如何合并两个或多个数组。

为了解决合并数组的问题，PHP 也给我们提供了一个好用的函数，那就是 array_merge()，其中，merge 意思就是合并。它的格式如下：

```
array_merge ($arrayName1, $arrayName2, …);
```

这个函数的变量要求非常的简单，直接将需要合并的数组罗列起来就可以了。但是凡事都有两面，看似简单的东西，实则不一定简单。这个 array_merge()函数就是这样：虽然结构简单，但运行的结果却有时让人摸不着头脑。我们一起来看一段脚本。

【例 6.14】 合并数组。

```
<?php
    $array1 = array("a", "b" => "b");
    $array2 = array("A", "b" => "B");

    $mergedArray = array_merge($array1, $array2);
    echo "<pre>";
    var_dump($mergedArray);
    echo "</pre>";
?>
```

在脚本的开始，定义了两个数组$array1 和$array2。数组$array1 包含两个元素，其中第二个元素的索引为“b”，值为“b”；数组$array2 也包含两个元素，其中第二个元素的

索引为“b”，值为“B”。接着我们使用 array_merge()函数将这两个数组合并在了一起并将合并后的数组赋值给了$mergedArray，最后我们使用 var_dump()函数输出了数组$mergedArray 的结构。大家可以推测一下这个脚本的执行结果，然后再看看图 6-14 所示，检查一下与你推测的结果是否相同呢？

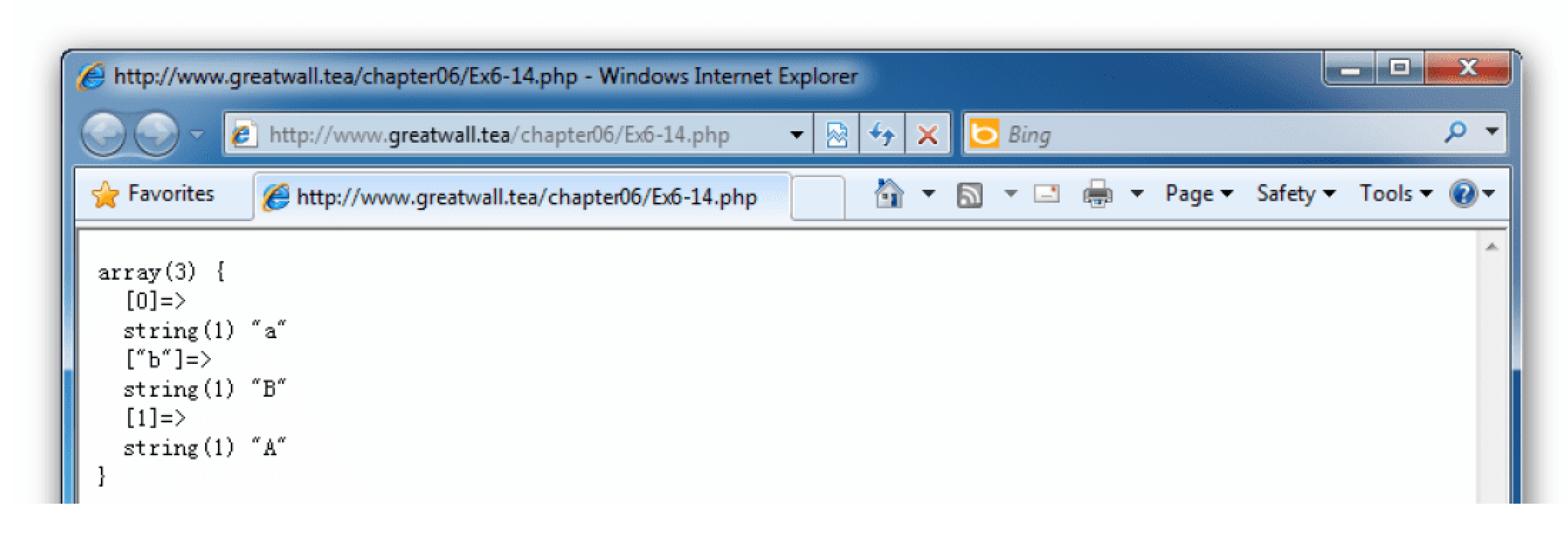


图 6-14　合并数组

看到这个结果，也许有的同学会欢呼，有的同学要叹气了。合并后的数组一共只有三个元素，其中数组$array1 和$array2 中的第一个元素都得到了保留，而只有数组$array2 中的第二个元素得到了保留。这是为什么呢？

PHP 处理引擎在执行这段脚本的时候，会轮流比对两个数组的每个元素。当它发现数组$array1 和$array2 中的第一个元素的索引相同时且均为数字索引时，它就会再为在新数组中为数组$array2 的第一个元素重新编号并将其置于新数组的末尾。当它发现数组$array1 的第二元素的索引为字符串且与数组$array2 的第二元素的索引相同时，它就会直接用数组$array2 的第二元素的值覆盖数组$array1 的第二元素的值并保留这个相同的索引。

简单地来说，相同数字索引的元素会被重新索引，使得这些元素的值均得以保留；而对于相同字符串索引的元素，这些元素值只有一个会被保留下来且使用其原始索引。

现在把 array_merge()函数中两个数组的位置换一下，结果又是什么呢？大家可以再进行推测一番。

最后，简单总结一下这一节中学到的两个函数如表 6-4 所示：

表 6-4　拆分与合并数组

拆分与合并函数	效 果 说 明
array_slice($arrayName, offset, subArray_len)	通过指定数组，子数组首元素相对母数组首元素的偏离值和子数组中元素的个数来截取母数组中的若干元素生成子数组
array_merge($arrayName1, $arrayName2, …)	通过指定若干数组，生成不含重复元素值的合并数组

6.5　多 维 数 组

看到“多维”这个词，大家可能就会想，难不成我们之前讨论的数组还只是数组的一种吗？是的。之前，我们讨论的是数组中的一维数组。说白了，就是只有一层索引的数组。

这里的多维指的是有多层索引的数组。

在 PHP 脚本中，我们可以定义数组索引层数是没有限制的。不过，如果定义的数组的索引超过了两层，就得考虑修改脚本了。简单地说，在编写 PHP 脚本时，如果需要使用多维数组的话，最多只用三维数组就好。

6.5.1　多维数组 vs. 一维数组

为了说清楚“多维数组”和“一维数组”之间的区别，我们先来看一张超市小票，如图 6-15 所示。

Purchased	Price
Apples	1.29
Pears	1.69
Asparagus	2.19
Grapes	1.85
Lettuce	0.87
Strawberries	2.79
Broccoli	1.06
Spinach	1.47
Potatoes	2.49
Tomatoes	1.35
Carrots	2.29
Peas	1.34
Clementines	2.79

图 6-15　超市小票

这张小票中反映出来的信息如果用一个一维数组来表示，就是如下的样子：

```
$purchased[apples] = 1.29;
$purchased[pears] = 1.69;
$purchased[asparagus] = 2.19;
$purchased[grapes] = 1.85;
$purchased[lettuce] = 0.87;
$purchased[strawberries] = 2.79;
$purchased[broccoli] = 1.06;
…
```

通过这个一维数组，我们可能很方便地查找和输出某件商品的价格。但是如果这张小票上的商品有数百多种，这种查找和输出是十分消耗服务器资源的。为了避免这种情况，我们再来看看这张小票，仔细找找其中的内在规律。

心细的同学也许已经发现了，这张小票上不是水果，就是蔬菜。那么可以把它们分成两类，然后建立一个二维数组，也就是如下的样子：

```
$purchased[fruit][apples] = 1.29;
$purchased[fruit][pears] = 1.69;
$purchased[vegetable][asparagus] = 2.19;
$purchased[fruit][grapes] = 1.85;
$purchased[vegetable][lettuce] = 0.87;
$purchased[fruit][strawberries] = 2.79;
$purchased[vegetable][broccoli] = 1.06;
…
```

我们也可以把这个二维数组看成是由数组组成的数组，它的结构可以被看做是如表 6-5 所示的样子：

表 6-5　二维数组的结构

	Key	Value	
		Key	Value
Purchased	fruit	apples	1.29
		pears	1.69
		grapes	1.85
		strawberries	2.79
		clementines	2.79
	vegetable	asparagus	2.19
		lettuce	0.87
		broccoli	1.06
		spinach	1.47

通过这张表，可以清楚地看到这个二维数组$purchased 是由两个一维数组$fruit 和$vegetable 组成的，每个一维数组又含有若干个元素。

6.5.2　创建多维数组和查看数组结构

在上一小节中，通过分析一张超市小票，理解了二维数组和一维数组的区别。那就是，二维数组可以被看做是由多个一维数组组成的。简单来说，多维数组和一维数组的区别就在于前者是以数组为元素的，而后者是以变量为元素的。

创建一个多维数组，既可以像上一节里那样，一个一个地定义数组中的每个变量，也可以使用 array()函数，就像创建一维数组一样。

如果用 array()函数来创建一个二维数组用于反映图 6-15 所示的这张超市小票的内容的话，脚本应该是如下的样子：

```
$purchased = array ("fruit" => array("apple" => 1.29, pears => 1.69, …),
"vegetable" => array("asparagus" => 2.19, lettuce => 0.87, …));
```

在这段脚本中，使用了三次 array()函数，第一次使用 array()函数创建了数组$purchased，定义了两个元素 fruit 和 vegetable，第二次和第三次使用 array()函数分别将 fruit 和 vegetable 定义为数组并向两个数组中添加了若干元素。

如果我们用 var_dump()函数输出数组$purchased 的结构，就应该是如图 6-16 所示的

样子。

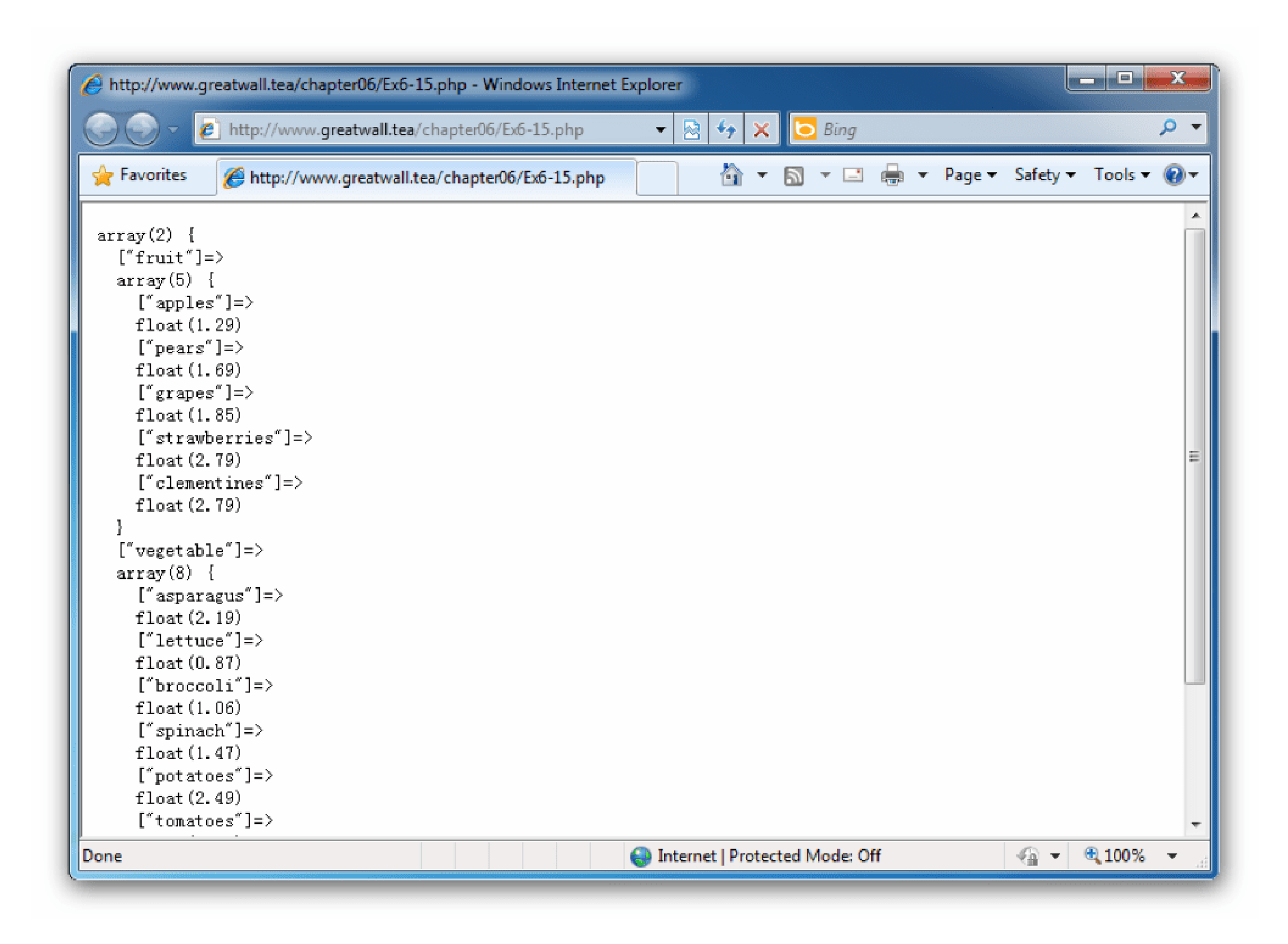


图 6-16　输出二维数组的结构

输出这个二维数组结构的原始脚本如下。

【例 6.15】 输出二维数组的结构。

```
<?php
   $purchased = array("fruit"     => array( "apples" => 1.29,
                                     "pears" => 1.69,
                                     "grapes" => 1.85,
                                     "strawberries" => 2.79,
                                     "clementines" => 2.79),
                   "vegetable" => array("asparagus" => 2.19,
                                     "lettuce" => 0.87,
                                     "broccoli" => 1.06,
                                     "spinach" => 1.47,
                                     "potatoes" => 2.49,
                                     "tomatoes" => 1.35,
                                     "carrots" => 2.29,
                                     "peas" => 1.34));

    //display the structure of $purchased
    echo "<pre>";
    var_dump($purchased);
    echo "</pre>";
?>
```

6.5.3　如何遍历多维数组

遍历多维数组与遍历一维数组的方法是类似的。我们可以使用在 6.2.1 小节中讲到的

foreach 语句来遍历多维数组。值得注意的是，foreach 语句是支持嵌套使用的。也就是说，一个数组有几层，我们就得嵌套几层 foreach 语句。

还记得 foreach 语句的结构吗？还是先来回顾一下：

```
foreach($arrayName as $key => $value);
]
```

如果需要嵌套使用的话，就变成了下面的样子：

```
foreach($arrayName as $upperLayer )
{
    foreach( $upperLayer as $lowerLayerKey => LowerLayerValue)
    {
        //在这里可以对数组元素进行一些操作
}
}
```

现在我们用 foreach 语句来输出上一小节中定义的二维数组$purchase，脚本如下。

【例 6.16】 遍历二维数组。

```
1   <?php
2    $purchased = array("fruit"    => array( "apples" => 1.29,
3                                  "pears" => 1.69,
4                                  "grapes" => 1.85,
5                                  "strawberries" => 2.79,
6                                  "clementines" => 2.79),
7                    "vegetable" => array("asparagus" => 2.19,
8                                  "lettuce" => 0.87,
9                                  "broccoli" => 1.06,
10                                 "spinach" => 1.47,
11                                 "potatoes" => 2.49,
12                                 "tomatoes" => 1.35,
13                                 "carrots" => 2.29,
14                                 "peas" => 1.34));
15
16       //遍历数组$purchased
17       foreach ($purchased as $category){
18           foreach ($category as $food => $price) {
19               $f price = sprintf("%01.2f", $price);
20               echo "$food: \$$f price <br>";
21           }
22       }
23  ?>
```

在 foreach 语句中，PHP 处理引擎是如何动作的呢？我们一起来看一看：

- ❑ PHP 处理引擎首先读取数组$purchased 的第一个元素（键值对），然后把这个元素的值存入变量$category 中。需要注意的是这个数组$purchased 的第一个元素是一个数组，于是变量$category 变成了数组$category。（第 17 行）
- ❑ 然后 PHP 处理引擎读取数组$category 的第一个元素（键值对），然后把这个元素的索引存入变量$food 中，把其对应的值存入变量$price 中。（第 18 行）
- ❑ 接着 PHP 处理引擎按照事先规定好的格式将价格格式化。（第 19 行）
- ❑ 紧接着使用 echo 语句输出一行记录。（第 20 行）
- ❑ 当内嵌的 foreach 语句在数组$category 中找不到可以读取的内容时，内嵌 foreach 语句循环结束，返回到上一层的 foreach 语句。这时 PHP 处理引擎会读取$purchased 的第二个元素（键值对）并将这个元素的值存入数组$category 中。（第 17 行）

- ❑ 然后 PHP 处理引擎读取数组$category 中的第一个元素（键值对），然后把这个元素的索引存入变量$food 中，把其对应的值存入变量$price 中。（第 18 行）
- ❑ 然后按照事先规定好的格式格式化价格，并输出记录直至结束。（第 19、20 行）
- ❑ 当外部 foreach 语句找不到可以读取的内容时，循环结束。（第 22 行）

简单地来说，外部的 foreach 语句将读取到的第一个元素$fruit 的值存入$category 中，然后内嵌的 foreach 语句开始遍历$category。当数组$category 遍历完成时，外部的 foreach 语句开始读取第二个元素$vegetable，并将读取到的值再次存入$category 中。接着内嵌的 foreach 语句开始遍历$category，直至循环结束。

6.6　实战练习：级联下拉菜单

6.6.1　界面预览

在本小节里我们利用所学的知识实现如图 6-17 所示的级联下拉菜单。当在第一个下拉菜单中选择了一个值时，第二个下拉菜单里的内容会随之发生变化。

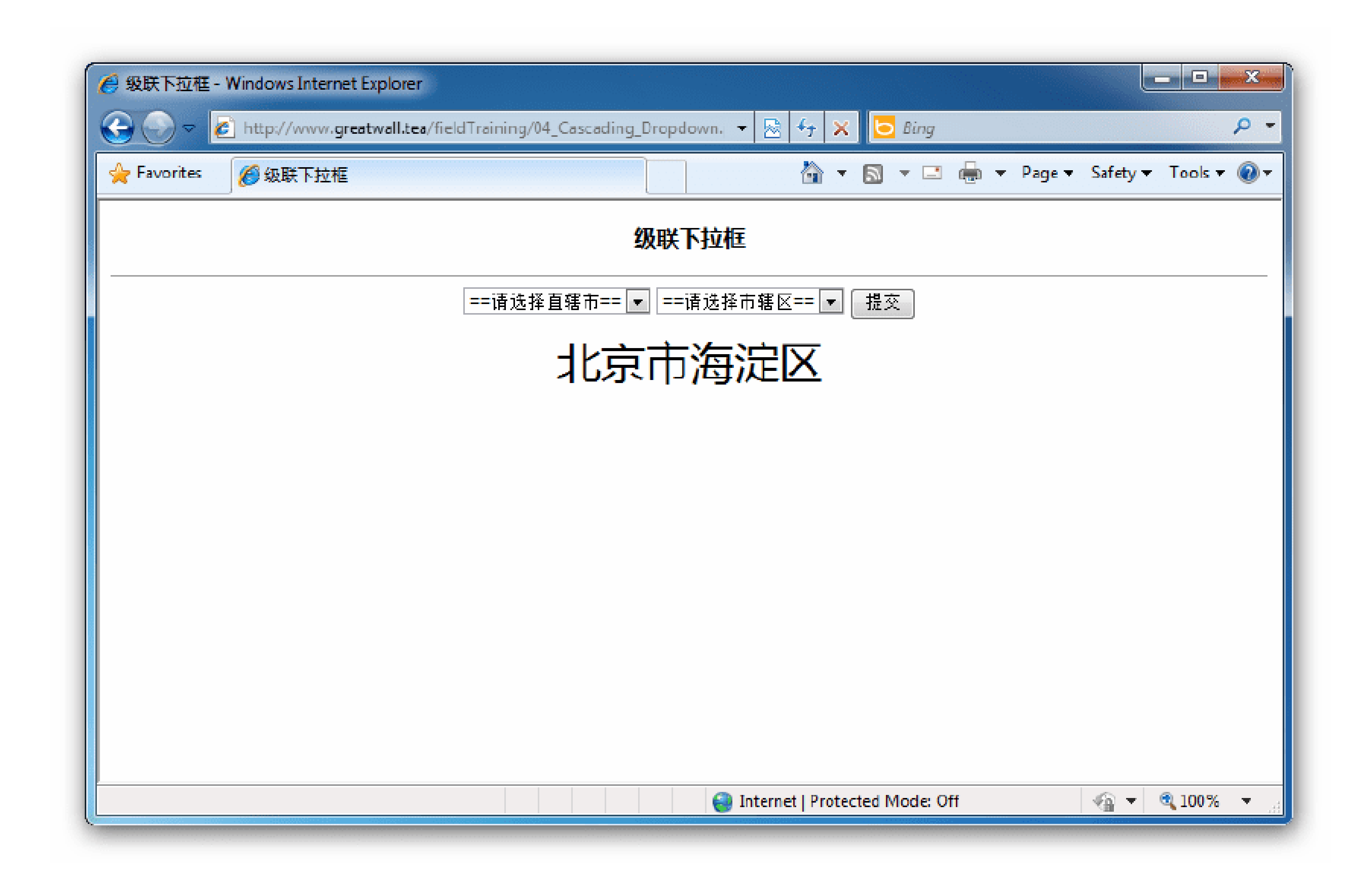


图 6-17　级联下拉框

在这个练习中，需要重点关注一下数组的拆分以及数组元素的遍历。例子的实现过程中涉及到 HTML DOM 和 JavaScript 的相关知识，已通过注释进行了解释。如果还有不明白的地方，可以在网上查找相关资料。

6.6.2　实现过程

首先，我们还是来看看 HTML 部分的代码：

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>级联下拉框</title>

        <style>
            body {
                font-size:small;
                font-family:'Microsoft YaHei';
            }

            form {
                text-align:center;
            }

            h3 {
                text-align:center;
            }

            #info {
                font-size:xx-large;
                line-height:60px;
                text-align:center;
            }

        </style>
    </head>
    <body>
        <h3>级联下拉框</h3>
        <hr />
        <form action='?check' method="post">
            <select name="prov" id="prov"></select>
            <select name="city" id="city"></select>
            <button type="submit">提交</button>
        </form>

        <div id="info"></div>       //在用户提交了表单后，展示用户选择的结果

        /* 在下面这段 JavaScript 脚本中，我们使用了 getElementById 将两个下拉列表
           框分别赋值给名为 prov 和 city 的对象，然后通过该对象的 add 方法向对象中添加
           列表项。在后续的 PHP 脚本中，我们也反复使用到了这段 JavaScript 脚本*/

        <script language="JavaScript">
            var prov = document.getElementById("prov");
            prov.options.add(new Option("==请选择直辖市==",'='));
            var city = document.getElementById("city");
            city.options.add(new Option("==请选择市辖区==",'='));
        </script>

    </body>
</html>
```

在 HTML 部分中，除了那段 JavaScript 脚本之外，与我们在之前的章节中看到的内容没有什么不同。现在再来看看 PHP 的部分：

```
<?php
    $mulicipality = explode(',','北京,天津,上海,重庆');
    $beijing = explode(',','东城区,西城区,海淀区,朝阳区,丰台区,石景山区,通州区,
```

```
顺义区,房山区,大兴区,昌平区,怀柔区,平谷区,门头沟区,密云县,延庆县');
$tianjin = explode(',','和平区,河西区,南开区,河东区,河北区,红桥区,东丽区,津
南区,西青区,北辰区,滨海新区,武清区,宝坻区,蓟县,宁河县,静海县');
$shanghai = explode(',','黄浦区,徐汇区,长宁区,静安区,普陀区,闸北区,虹口区,
杨浦区,闵行区,宝山区,嘉定区,浦东新区,金山区,松江,青浦区,奉贤区,崇明县');
$chongqing = explode(',','渝中区,大渡口区,江北区,沙坪坝区,九龙坡区,南岸区,
北碚区,綦江区,双桥区,渝北区,巴南区,万州区,涪陵区,黔江区,长寿区,江津区,合川区,永
川区,南川区');

/* 在上面的五句中，我们使用 explode()函数将五串由逗号分隔的字符串转换成了数组。
   其中，第一个数组$mulicipality 中的内容将填充到如图 6-17 所示的左边的下拉框中，
   而余下四个数组则会根据用户在左边下拉框中的选择，填充到右边的下拉框中。      */

$bjStr = '';
$tjStr = '';
$shStr = '';
$cqStr = '';

//上面这四个变量是用来存放处理成 JavaScript 脚本的数组元素

foreach ($mulicipality as $key => $value) {
    echo '<script>prov.options.add(new Option("'.$value.'","'.$key.
    '"));</script>';
}
//上面这段脚本用来遍历数组$mulicipality，并将其键和值拼合到 JavaScript 脚本中

foreach ($beijing as $key => $value) {
    $bjStr .= 'city.options.add(new Option("'.$value.'","'.$key.
    '"));';
}

foreach ($tianjin as $key => $value) {
    $tjStr .= 'city.options.add(new Option("'.$value.'","'.$key.
    '"));';
}

foreach ($shanghai as $key => $value) {
    $shStr .= 'city.options.add(new Option("'.$value.'",
    "'.$key.'"));';
}

foreach ($chongqing as $key => $value) {
    $cqStr .= 'city.options.add(new Option("'.$value.'",
    "'.$key.'"));';
}

//上面这四段脚本用来遍历剩余的四个数组，并将它们的键和值拼接到对应的 JavaScript
   脚本中

echo '<script language="JavaScript">       //从此往下为 JavaScript 脚本
    prov.onchange = function () {
                    //当 prov 下拉列表框的值发生改变时，会触发如下函数
    selectedProv = prov.options[prov.selectedIndex].value;
                    //获取用户的选择
    switch(selectedProv){
        case "0":
            removeAll();            //移除当前 city 下拉列表框中的所有列表项
            city.options.add(new Option("==请选择市辖区==","="));'.
```

```
                                //向 city 下拉框中添加列表头
            $bjStr.             //用 PHP 向 city 下拉框中添加北京市的市辖区
            'break;
        case "1":
            removeAll();
            city.options.add(new Option("==请选择市辖区==","="));'.
            $tjStr.             //用 PHP 向 city 下拉框中添加天津市的市辖区
            'break;
        case "2":
            removeAll();
            city.options.add(new Option("==请选择市辖区==","="));'.
            $shStr.             //用 PHP 向 city 下拉框中添加上海市的市辖区
            'break;
        case "3":
            removeAll();
            city.options.add(new Option("==请选择市辖区==","="));'.
            $cqStr.             //用 PHP 向 city 下拉框中添加重庆市的市辖区
            'break;
    }
}

function removeAll() {
    for (i=0;i<20;i++){
        city.options.remove(city.options[i]);
                        //这里使用了 city 对象的 remove 方法移除指定的对象
    }
}
</script>';

/* 拼合 JavaScript 脚本，并输出。在上面这段脚本中，我们定义了在如图 6-7 所示的页
   面中，当用户在左边的下拉框中选择了某个值时，浏览器会记录用户选择的列表项对应的
   值，然后根据获取到的值向右边的列表框中添加列表项                            */

if(isset($_REQUEST['check'])){      //判断用户是否提交了表单
    $prov = $_REQUEST['prov'];      //获取用户选择的直辖市
    $city = $_REQUEST['city'];      //获取用户选择的市辖区

    if($prov == '=' || $city == '='){       //判断用户是否做出了选择
        $msg = '请选择直辖市和市辖区';
    } else {
        switch ($prov) {                    //将用户的选择转换成可读的文字
            case '0':
                $msg = '北京市'.$beijing[intval($city)];
                break;
            case '1':
                $msg = '天津市'.$tianjin[intval($city)];
                break;
            case '2':
                $msg = '上海市'.$shanghai[intval($city)];
                break;
            case '3':
                $msg = '重庆市'.$chongqing[intval($city)];
                break;
        }
    }
    //向 ID 为 info 的 DIV 输出用户的选择
```

```
        echo '<script language="javascript">document.
        getElementById("info").innerHTML="'.$msg.'"</script>';
    }
    ?>
```

上面这段脚本一共完成实现了如下两个功能：

（1）在用户选择了一个直辖市后，其右边的市辖区的内容也会发生变化。

（2）在用户提交所选城市和市区后，在表单的下方会以粗体展示用户所选。

下面来看看上面的代码是如何实现这两个功能的。

功能 1　在用户选择了一个直辖市后，其右边的市辖区的内容也会发生变化。

首先我们定义了五个变量，分别为$municipality、$beijing、$tianjin、$shanghai 和$chongqing 用来存储这四个城市和每个城市所辖地区的名称。这些变量都是使用 explode()函数生成的数组变量。

接下来，定义了四个空变量，用来存放 JavaScript 语句。需要注意的是，在一个 PHP 文件中，只有 PHP 脚本和非 PHP 脚本之分，我们可以在任何非 PHP 脚本中通过“<?php”和“?>”来插入 PHP 脚本。以$bjStr 为例，我们使用 foreach 语句对$beijing 这个数组进行遍历，并将遍历的值混合到 JavaScript 的数组语句中，如下所示：

```
$bjStr .= 'city.options.add(new Option('".$value."','".$key."'));';
```

在这条语句中，我们使用“.”操作符连接了 city.options.add(new Option(' ".$value. " ', '".$key." '))三个字符串以及$value 和$key 两个 PHP 变量的值。其中，“city.options.add”中的“city”是我们在 HTML 中定义了一个下拉框。另一个下拉框为“prov”；“options”是下拉框元素的子元素；而“add”则是向下拉框中添加子元素的方法。这些都是 JavaScript 语句，在这里就不展开了。

在定义好$bjStr、$tjStr、$shStr 和$cqStr 之后，我们就开始向 HTML 页面中添加 JavaScript 代码。在这里定义了“prov”下拉框的 onchange()函数，当“prov”下拉框的值发生变化时，就会加载相应城市的所辖区。代码如下：

```
echo '<script language="JavaScript">        //从此往下为 JavaScript 脚本
        prov.onchange = function () {
                        //当 prov 下拉列表框的值发生改变时，会触发如下函数
        selectedProv = prov.options[prov.selectedIndex].value;
                        //获取用户的选择
        switch(selectedProv){
            case "0":
                removeAll();            //移除当前 city 下拉列表框中的所有列表项
                city.options.add(new Option("==请选择市辖区==","="));'.
                                        //向 city 下拉框中添加列表头
                $bjStr.                 //用 PHP 向 city 下拉框中添加北京市的市辖区
                'break;
            case "1":
                removeAll();
                city.options.add(new Option("==请选择市辖区==","="));'.
                $tjStr.                 //用 PHP 向 city 下拉框中添加天津市的市辖区
                'break;
            case "2":
                removeAll();
                city.options.add(new Option("==请选择市辖区==","="));'.
                $shStr.                 //用 PHP 向 city 下拉框中添加上海市的市辖区
```

```
                'break;
            case "3":
                removeAll();
                city.options.add(new Option("==请选择市辖区==","="));'.
                $cqStr.      //用 PHP 向 city 下拉框中添加重庆市的市辖区
                'break;
        }
    }

    function removeAll() {
        for (i=0;i<20;i++){
            city.options.remove(city.options[i]);
                            //这里使用了 city 对象的 remove 方法移除指定的对象
        }
    }
</script>';
```

在这段向 HTML 页面中添加内容的 JavaScript 代码中，我们同样是使用了“.”操作符连接了 JavaScript 代码和 PHP 变量。值得注意的是，在这段 JavaScript 代码中，还定义了一个 removeAll()函数，用来在加载用户所选城市辖区的同时，移除之前加载的城市辖区。

这样，我们就实现了第一个功能。

功能 2　在用户提交所选城市和市区后，在表单的下方会以粗体展示用户所选。

为了实现这个功能，我们做了两件事情。第一件事，就是在 HTML 页面的表单中定义了“action”和“method”两个属性。其中“action”属性值为“?check”，而“method”的值为“post”。前者定义了处理用户提交数据的页面，而后者则定义了向该页面传递用户提交数据的方式。关于如何在页面间传递数据，大家可以参考 9.1 节的内容。

这里大家只要知道，我们将用户提交的数据传递到了当前页面，并通过如下的脚本来判断和显示用户提交的内容就可以了。

```
if(isset($_REQUEST['check'])){          //判断用户是否提交了表单
        $prov = $_REQUEST['prov'];      //获取用户选择的直辖市
        $city = $_REQUEST['city'];      //获取用户选择的市辖区

        if($prov == '=' || $city == '='){       //判断用户是否做出了选择
            $msg = '请选择直辖市和市辖区';
        } else {
            switch ($prov) {            //将用户的选择转换成可读的文字
                case '0':
                    $msg = '北京市'.$beijing[intval($city)];
                    break;
                case '1':
                    $msg = '天津市'.$tianjin[intval($city)];
                    break;
                case '2':
                    $msg = '上海市'.$shanghai[intval($city)];
                    break;
                case '3':
                    $msg = '重庆市'.$chongqing[intval($city)];
                    break;
            }
        }
        //向 ID 为 info 的 DIV 输出用户的选择
        echo '<script language="javascript">document.getElementById
        ("info").innerHTML="'.$msg.'"</script>';
```

```
    }
    ?>
```

在这段脚本里，我们使“isset($_REQUEST['check'])”来判断用户是否提交了表单。这里的“$_REQUST[]”是个预定义数组，用来存放用户通过 URL 或表单传递的数据。如果“isset($_REQUEST['check'])”的结果为真，则表示用户提交了表单；反之，则没有提交表单。

如果判断当前用户提交了表单，则获取用户通过表单提交的直辖市和市辖区，并将这两个信息存入变量$prov 和$city 中。由于我们通过 JavaScript 生成的下拉框中的值和名分别为之前定义的$minucipality、$beijing、$tianjin、$shanghai 和$chongqing 这五个数组各元素的索引和值，所以用户通过表单传递过来的只是所选直辖市和市辖区的索引值，因此我们需要根据索引值，再回到上述五个数组中查找对应的直辖市和市辖区的名字。另外，需要注意的是，用户通过表单传递过来的值，无论其是数字还是字符，其数据类型都是字符，因此，我们使用了“intval()”这个函数转换了一下数据类型，以便在数组中查找用户通过表单传递过来的索引对应的城市和市区名称。

将查到了用户通过表单传递过来的索引对应的城市和市区名称后，我们使用一段 JavaScript 代码将获取到的内容输出到 HTML 页面中定义的 ID 为“info”的 DIV 元素中。

这样，我们就实现了第二个功能。

6.7　习　　题

（1）请判断下面脚本的运行结果：

```
<?php
    $a = array("a" => 5, "b" => "banana", "c" => "egg");
    $b = array(10, "apple","pie");
    $c = array merge($a,$b);
    $d = array slice($a,0,2);
    echo "<pre>";var dump($c);echo "</pre>";
    echo "<pre>";var dump($d);echo "</pre>";
?>
```

（2）请使用手动遍历的方法输出上面脚本中数组$a 的所有元素。

（3）请定义一个包含 10 个元素的数组，然后使用 foreach 语句遍历这个数组。

（4）请使用 implode()函数合并你定义的数组为一个字符串。

（5）请使用 explode()函数将上一题中的字符串转换成一个数组。

第 3 篇　PHP 编程基础

第 7 章　条件与循环

在之前的章节中，除了第 6 章使用了 foreach 语句的脚本之外，其他的脚本都是直线型的（不包括实战练习中的脚本示例）。PHP 处理引擎会按照从上到下的顺序执行脚本中的每一行。如果想要调整某脚本中语句的执行顺序，直接将需要提前执行的语句放到前面就可以了。

如果世界总是如此的简单，那该有多美好啊，可惜这种美好只存在于理想世界中。在现实世界里，我们会有着无穷无尽的目标需要去达成，只有满足了某些条件，我们才能实现这些目标。比如，如果想要涨工资，就得好好干活，只有业绩达标了，才有可能给你涨工资。再比如，如果想要拿奖学金，就得埋头苦干地好好学习，只有学习成绩上去了，才有希望拿奖学金。另一方面，要想业绩达标，就得日复一日地重复着手头的工作；要想提升学习成绩，就得年复一年地读书考试。所谓的做一天和尚撞一天钟，讲的就是这个道理。

几乎所有的计算机语言都是为了让计算机程序替人们做些简单的判断以及模拟人的某些重复性的活动，从而把人从繁重的劳动中解放出来。这听上去有些像是在讲马克思主义政治经济学了。那我们言归正传，看看 PHP 是如何通过脚本模拟这些活动的吧。

7.1　精细化运算——条件

我们还是用在第 3 章里提到的一个例子来说说什么是条件。这个例子是这样的：

```
1   if (it is morning)
2   {
3       get up;
4       brush my teeth;
5       wash my face;
6       put on my jacket;
7       go to work;
8   }
```

在这个例子中，第 1 行定义了一个条件。只有当这个条件得到满足时，才会执行第 3 到第 7 行的内容。换句话说，只有当前时间是早晨时，我们才会进行诸如起床、刷牙、洗脸、穿衣、赶着去上班这些活动。如果当前时间是下午，我们就有另外一套动作了。

再来看一个例子。不知道大家有没有注意到，当我们成功登录了腾讯 QQ 邮箱后，会在邮箱首页上看到如图 7-1 所示的欢迎辞，这个欢迎辞还是会随着时间发生变化的：早上登录的时候，它会说“早上好”，中午登录的时候，它会说“中午好”。晚上登录的时候，它会说“晚上好”。

实现这个功能其实非常简单。我们可以通过定义一些条件，让电脑输出不同的信息。

【例 7.1】 定义条件。

```
1   <?php
2       $currentHour = date("H");
3        if ($currentHour < 12) {
4            echo "Good morning!";
5        } elseif ($currentHour > 12 AND $currentHour <18){
6            echo "Good afternoon!";
7        } elseif ($currentHour >18 AND $currentHour < 24){
8            echo "Good evening!";
9        }
10  ?>
```



图 7-1　QQ 邮箱首页

例 7.1 中的脚本模拟了 QQ 邮箱首页的问候功能。首先，将当前时间的小时数赋值给变量$currentHour，然后使用条件语句定义了如下三个条件：

- $currentHour 小于 12；
- $currentHour 大于 12 且小于 18；
- $currentHour 大于 18 且小于 24。

通过比对当前时间与这些条件的匹配情况来要求 PHP 处理引擎输出不同的问候语。在脚本中，第 3 行、第 5 行和第 7 行括号里的内容就是用 PHP 语言对这三个条件的描述。下面就具体来看看如何定义条件。

7.1.1　什么是条件

在定义条件之前，需要知道什么是条件。

在 PHP 脚本中，条件是一种表达式，通常用在复杂脚本中。它的判断结果通常只有两

种，一种为“true”，另一种为“false”。PHP 引擎根据条件的判断结果来决定脚本中某些部分是否需要执行。定义一个条件，需要比较两个或多个值的大小。在比较值的大小时，通常会问到的问题包括：

- ❑ 两个值是否相等？比如，你喝的啤酒的品牌和我喝的是不是一样的？再比如，你的名字是不是叫金三顺？
- ❑ 两个值中，是不是一个比另一个大？比如，你的年纪是不是比我大？再比如，你买的这套衣服得 1000 多块钱吧？
- ❑ 某个字符串的格式是否与指定的样式匹配？比如，你是不是姓李？你们家是不是住在鄱阳街？

当然，我们可以一次定义多个条件，并指定这多个条件之前的关系。比如，你爸爸是不是在红旗厂工作，要不就是在晨光厂工作？再比如，你哥哥今年 15 岁，你妹妹今年也 12 岁了吧？前面两个问题之间的关系是“或”，而后面两个问题之间的关系是“和”。这也是定义多个条件时最常用的两种关系。

总结一下，所谓条件，其实就是简单疑问句，也就是可以用是或否来回答的问句。在 PHP 中使用条件，只需要把问题用脚本语言写出来就成了。那么应该怎么写呢？

在写之前，还是先来掌握一些定义条件时常用的“词汇”和“句式”吧。

7.1.2　如何定义条件

在本小节里，我们就按照上一小节中提到的三类问题来讲一讲如何定义条件吧。

1．值的比较

第一类和第二类问题讲的是值的比较。在比较的时候，会使用一些比较操作符来连接被比较的两个对象，如表 7-1 所示。

表 7-1　常见的比较操作符

操　作　符	含　　义
==	操作符前后的两个变量值是否相等
===	操作符前后的两个变量的类型和值是否都相等
>	操作符前的变量值是否比操作符后的变量值大
>=	操作符前的变量值是否大于等于操作符后的变量值
<	操作符前的变量值是否比操作符后的变量值小
<=	操作符前的变量值是否小于等于操作符后的变量值
!=, <>	操作符前后的两个变量的值是否不相等
!==	操作符前后的两个变量的类型和值是否都不相等

这些操作符可以连接的不仅仅是数字，也可以用于连接字符串。有的同学可能不理解，字符串怎么能比较大小呢？在 PHP 中，字符串的大小是有规定的。按照字母表的顺序，大写字母在前，小写字母在后。举个例子“SOS”就要比“So”大。标点符号之前也可以比较大小，只不过，我们通常不会这么干。因为比较标点符号的大小没有什么实际意义。

我们来看一些例子：

```
$a == $b
$age != 12
$ageU > $ageMe
$suitePrice > 1000
```

在上面的四个例子中，第一个例子比较特别，在比较两个变量的值是否相等的时候，用的是两个等号（==），而不是一个等号（=）。一个等号在 PHP 脚本里叫赋值符，比如“$a = 16”的意思是把 16 这个值赋给变量$a，而两个等号才是比较操作符，比如“$a == $b”的意思是变量$a 和变量$b 的值是否相等。两者的区别务必要牢记，不要用错了。

PHP 处理引擎在看到这些条件后，会对操作符前后的两个变量按照操作符的意义进行比较。如果比较的结果与操作符的定义相符，返回一个布尔值 true。如果比较的结果与操作符的定义不相符，则返回另一个布尔值 false。

比如，假设$a = 1，而$b = 2。那么当 PHP 处理引擎处理条件$a == $b 时，会返回 false，因为两个变量的值不相等。

除了使用操作符之外，我们还可以使用一些判断函数来进行判断。

PHP 提供了许多的判断函数，比较常见的有如下几种，如表 7-2 所示。

表 7-2　判断函数

判 断 函 数	含　　义
isset($*varName1*,$*varName2*,…)	判断指定对象是否已定义。若含有多个对象，则其中一个对象未定义，系统亦会返回 false
is_int($*varName*)	判断指定对象是否为整型变量
is_array($*varName*)	判断指定对象是否为数组
is_float($*varName*)	判断指定对象是否为浮点型变量
is_string($*varName*)	判断指定对象是否为字符串型变量
is_numeric($*varName*)	判断指定对象是否为数字或数字型字符串
is_null($*varName*)	判断指定对象的值是否为零

这些判断函数可能帮助我们很快地对一个或多个对象的类型进行判断，从而对这些对象做出相应的动作。我们也可以在这些判断函数前面加上“!”来做出相反的判断。比如“!isset()”可以用来判断指定的一个或多个对象是否未定义，“!is_string()”则可以用来判断指定对象是否不是一个字符串。

2．正则表达式

在上一小节中关于第三类条件讨论的其实就是模式匹配的问题。为了实现模式匹配，就得先定义一个模式，而定义模式就要用到正则表达式。接下来，简单地讨论一下正则表达式。

大家如果用过 MS-DOS 操作系统，那么一定用过类似于下面的命令：

```
C:\ dir ex*.txt
```

在 MS-DOS 操作系统中，这条命令可以打印出 C 盘根目录下所有的以“ex”开头的 TXT 文件。这个“*”叫通配符。正则表达式与通配符类似，但是前者稍微要复杂一些。

最常使用正则表达式的地方就是检测用户输入的内容是否符合要求。比如在中国，手机号码都是以 1 开头的，当声称是来自中国的用户输入了一串不是以 1 开头的数字就不能被存入数据库。再比如湖北省的身份证号码都是以 42 打头的，当声称是来自湖北省的用户

输入一串不是以 42 开头的字符串就也不能被存入数据库。

正则表达式中除了有英文字符之外，还有一些特殊字符。先来看看有哪些特殊字符？以及这些特殊字符的作用，如表 7-3 所示。

表 7-3　正则表达式中的特殊字符

特殊字符	含义	举例	匹配字串	不匹配字串
^	以……开始	^e	egress	ingress
$	以……结束	m$	ham	Hen
.	任一字符	..	up, great	A, 1
?	前一位字符可选	ger?m	gem, germ	gm
( )	必含字符	g(er)m	germ	gem, grem
[ ]	可选字符	g[er]m	gem, grm	germ, gel
[^]	不可选字符	g[^er]m	gym, gum	gem, grm, grem, germ
-	在前置和后置字符之间的所有字符	^[A-J]eep	Jeep, Beep	Keep
+	前置项目可多选	Shop[1-3]+	Shop1, Shop12, Shop321	Shop456
*	前置项目可重复	ge*m	gm, geeem	germ
{n}	前置项目必重复次数	ge{5}m	geeeeem	geem, geeem
{n1,n2}	前置项目可重复次数范围	ge{2,5}m	geem, geeem, geeeem, geeeeem	Gem
\	后置项目不含特殊含义	g*m	g*m	ggm, gem
(|)	可选字串	I want an (apple | orange).	I want an apple. I want an orange.	I want a banana.

表里的内容看上去是不是很复杂呢？下面有几道选择题，大家可以尝试解答一下，消化消化上面讲到的内容。

```
1.“一串以任意字母开头任意长度的字符串”可用正则表达式表示为：
a)  [A-Za-z].* b)  ^[A-Za-z].*c)  ^[A-Za-z]* d)  ^[A-Za-z].
```

在这一题中，需要定义一串以任意字母开头且长度不定的字符串，可以用到的特殊符号应该有：

- “^”标记起始位置；
- “[-]”包含从 A 到 Z 和 a-z 的所有字母；
- “.*”标记任意长度的字符串。

这样一分析后，答案就呼之欲出了，应该是 b。

```
2.下面哪一个正则表达式可以匹配“My Dear Kim”这个字符串？
a)  ^Dear (Kim | Jim)                    b)  Dear [Kim]$
c)  Dear (Kim | Jim)                     d)  Dear {Kim,Jim}
```

- 在这一题中的各个选项中，除了选项 a 用了“^”标示起始字符为“D”以外，其他各项都没有用起始位置符。而我们需要匹配的字符串的起始字符为 M。选项 a 排除。
- 选项 b 中使用了“[ ]”和“$”符标志可与其匹配的字符串只能是“… Dear K”、

"…Dear i"和"…Dear m"这三个字符串。选项 b 排除。

- ❑ 选项 c 使用了"(|)"标示二选一，与之匹配的字符串可以是"… Dear Kim"和"… Dear Jim"。于是 My Dear Kim 这个字符串是可以匹配该正则表达式的字符串之一。选项 c 正确。
- ❑ 选项 d 使用了"{ , }"标示范围，但用错了参数。这个特殊符号之间的参数必须为数字。选项 d 排除。

```
3.下面哪一个正则表达式可以匹配"9001"和"7851-32564"这样的两串字符？
a)  ^[0-9]{4}\-[0-9]{5}$              b)  ^[0-9]{4}(\-[0-9]{5})$
c)  ^[0-9]{4}(\-[0-9]{5})?$           d)  ^[0-9]{4}\-[0-9]{5}?$
```

"9001"和"7851-32564"这样的两串字符，前者是由四个数字组成的，可以写成"^[0-9]{4}$"。后者是由四个数字加个连字符再加五个数字组成的，可以写成^[0-9]{4}\-[0-9]{5}$，也就是选项 a。所以说选项 a 只能匹配"7851-32564"这样的字串，却不能匹配"9001"这样的字串。如果想要让正则表达式"^[0-9]{4}\-[0-9]{5}$"也匹配"9001"这样的字串，我们就得对其进行改造。

再来找找这两个字串的共同点："9001"是由四个数字组成，而"7851-32564"是由四个数字开头的，也就是说把连字符加后面的五个字符变成可选的就行了。选项 b 和选项 d 都有标示前置字符为可选的"？"。两者的区别就在于选项 b 中用了一对"( )"把连字符和五个数字组成的字串标示为必含字符了。这样一来，选项 c 的中"？"的作用范围就由五个数字变成了连字符加五个数字了。而选项 d 的中"？"的作用范围只是五个数字而已。至于选项 c 中的"( )"不过是强调这个字符串必须以连字符加五个数字结尾罢了。

```
4. 正则表达式"^.+@.+\.com$"可以匹配如下哪一个字符串？
a)  you@example.com                b)  1943@example.net
c)  @example.com                   d)  youremailaddress@example.org
```

来分析一下这个正则表达式：

- ❑ "^.+"表示以一个任一字符开头的任意长度的字符串，
- ❑ "@.+"表示一个接在"@"后的任意长度的字符串，
- ❑ "\.com$"表示一个以".com"结尾的字符串。

把它们三个连在一起，就是以任一字符开头，包含"@"，并以".com"结尾的任意长度的字符串。说白了它匹配的就是以".com"结尾的邮箱地址。而选项 b 和选项 d 都不是以".com"结尾的，可以排除了。而选项 c 就错在它是以"@"开头的。所以只有选项 a 才是正确的。

现在可以写一段脚本来验证一下我们的分析是否正确。在这段脚本中，会用到 ereg() 函数，它带有如下两个参数：

```
preg_match("pattern", $variable)
```

其中 pattern 指的是正则表达式，注意一定要用一对引号把正则表达式包裹起来。$variable 则代表指定的变量。用于检测的脚本如下：

【例 7.2】 检测字符串与正则表达式是否匹配。

```
$array1 = "Hello World!";
$array2 = "My Dear Kim";
$array3 = "9001";
```

```
4   $array4 = "7851-32564";
5   $array5 = " you@example.com";
6
7   if (preg_match("/^[A-Za-z].*/", $array1)) {
8       echo "$array1 matches the regular expression.";
9   } else {
10      echo "$array1 does not matches the regular expression.";
11  }
12
13  if (preg_match ("/Dear (Kim | Jim)/", $array2)) {
14      echo "$array2 matches the regular expression.";
15  } else {
16      echo "$array2 does not matches the regular expression.";
17  }
18
19  if (preg_match ("/^[0-9]{4}(\-[0-9]{5})$/", $array3)) {
20      echo "$array3 matches the regular expression.";
21  } else {
22      echo "$array3 does not matches the regular expression.";
23  }
24
25  if (preg_match ("/^[0-9]{4}(\-[0-9]{5})$/", $array4)) {
26      echo "$array4 matches the regular expression.";
27  } else {
28      echo "$array4 does not matches the regular expression.";
29  }
30
31  if (preg_match ("/^.+@.+\.com$/", $array5)) {
32      echo "$array5 matches the regular expression.";
33  } else {
34      echo "$array5 does not matches the regular expression.";
35  }
```

这段脚本实在太长了，而且重复的内容很多。如果我们把所有的条件都放在一起就好了。

其实在 7.1.1 小节中，我们就提过，可以一次性定义多个条件，然后用“和”或“或”把这些条件连接起来。用脚本应该怎么写呢？

在 PHP 脚本中，我们可以使用 and、or 或 xor 来连接两个条件。其中：

- ❑ and 表示两个条件均为真（true）时，复合条件为真；
- ❑ or 表示两个条件中任一条件为真或两个条件均为真（true）时，复合条件为真；
- ❑ xor 表示两个条件中任一条件为真（true）时，复合条件为真。

使用这些连字符，我们可以把例 7.2 所示脚本的第 7 行到第 35 行的代码改成如下的样子：

```
7   if (preg_match("/^[A-Za-z].*/", $array1) &&
8     preg_match ("/Dear (Kim | Jim)/", $array2) &&
9     preg_match ("/^[0-9]{4}(\-[0-9]{5})$/", $array3) &&
10    preg_match ("/^[0-9]{4}(\-[0-9]{5})$/", $array4) &&
11    preg_match ("/^.+@.+\.com$/", $array5))
12  {
13      echo "These arrays match their corresponding regular expressions.";
14  } else {
15      echo "Either certain or all arrays do not match their corresponding
regular expressions.";
16  }
```

在这短短十行的代码里，我们使用 and 把这五个 preg_match()函数连接了起来。也就是说，只有当这五个 ereg()函数返回的值均为 true 时，才会打印如下这句话：

```
These arrays match their corresponding regular expressions.
```

否则，系统会输出：

```
Either certain or all arrays do not match their corresponding regular expressions.
```

7.1.3　简单条件语句 if…else…

通过前两个小节的讲解，大家对这个单条件语句肯定已经不陌生了。在本小节里，我们将系统地来讲讲这个单条件语句。它的结构如下。

【例 7.3】 简单条件语句 if… else… 的结构。

```
if (condition1) {
    statement block A
} elseif (condition2) {
    statement block B
} else {
    statement block C
}
```

如例 7.3 所示，简单条件语句 if 由三个部分构成。如果语句块 A、语句块 B 和语句块 C 中都只有一条语句的话，那么花括号也是可以去掉的。

（1）if：这一部分是必选的，用于指定判断条件。

- 如果条件的判断结果为 true：系统执行语句块 A 中语句。在语句块 A 中的语句执行完毕后，系统将直接执行语句块 C 之后的内容，语句块 B 和语句块 C 中的语句被全部忽略了。
- 如果条件的判断结果为 false：语句块 A 中的语句则被忽略，系统将转而执行紧随其后的语句。需要说明的是，紧随其后的可以是 elseif 语句、else 语句或其他的语句。

（2）elseif：这一部分是可选的，用于指定另一个判断条件。

- 如果条件的判断结果为 true：系统执行语句块 B 中的语句。在语句块 B 中的语句执行完毕后，系统将直接执行语句块 C 之后的内容。语句块 C 中的语句则被忽略了。
- 如果条件的判断结果为 false：语句块 B 中的语句则被忽略。系统将转而执行紧随其后的其他语句。需要说明的是，紧随其后的可以是另一个 elseif 语句、else 语句或其他语句。

（3）else：这一部分也是可选的。只有当它前面的一个条件判断为 false 时，这一部分中包含的语句才会被执行。在例 7.3 中，只有当 elseif 中的条件判断为 false 时，语句块 C 中的语句才会被执行。一个条件语句只能含有一个 else 部分。如果在一个条件语句中使用了 else 部分，该条件语句必须以 else 部分作结束。

现在来看一个例子。

假设你是一位老师，需要将 100 名学生的百分制成绩转换成五级制，并将转换后的成绩输出到浏览器告知学生。我们应该怎么做呢？

【例 7.4】 简单条件语句示例。

```
<?php
    $actualScore = 87;
```

```
3
4       if ($actualScore > 92){          //92 分以上
5           $grade = "A";
6           $message = "Well Done!";
7       } elseif ($actualScore <= 92 and $actualScore > 80) {
    //80~92（含 92）分
8           $grade = "B";
9           $message = "Good! Keep on trying!";
10      } elseif ($actualScore <= 80 and $actualgrade > 70) {
    //70~80（含 80）分
11          $grade = "C";
12          $message = "Okay! Not bad!";
13      } elseif ($actualScore <= 70 and $actualScore > 60) {
    //60~70（含 70）分
14          $grade = "D";
15          $message = "Uh oh! You've just won the game!";
16      } else {                                       //60 及 60 分以下
17          $grade = "F";
18          $message = "Try harder next time!";
19      }
20
21      echo "$message<br>";
22      echo "You grade is $grade.";
23  ?>
```

在例 7.4 这段脚本中，定义了一个变量$actualScore 用于存放学生的实际分数，然后将实际分数与转换标准做比较：

- 当实际分数高于 92 分时，学生的等级为 A，评语为“Well Done!”。这时系统会略过第 7 到第 20 行的内容，直接执行第 21 行：输出评语和等级。
- 当实际分数低于 92 分但高于 80 分时，学生等级为 B，评语为“Good! Keep on trying!”。这时系统会略过第 5 行、第 6 行以及第 10 到第 20 行，直接执行第 21 行：输出评语和等级。
- 当实际分类低于 80 分但高于 70 分时，学生等级为 C，评语为“Okay! Not bad!”。这时系统会略过第 5 到第 9 行以及第 13 到第 20 行，直接执行第 21 行：输出评语和等级。
- 当实际分类低于 70 分但高于 60 分时，学生等级为 D，评语为“Uh oh! You've just won the game!”。这时系统会略过第 5 到第 12 行以及第 16 到第 20 行，直接执行第 21 行：输出评语和等级。
- 当实际分数低于 60 分时，学生的等级为 F，评语为“Try harder next time!”。这时系统会略过第 5 到 15 行，直接执行第 21 行：输出主语和等级。

在例 7.4 中，假定某位学生的实际成绩为 87 分。那么执行的结果应该如图 7-2 所示。

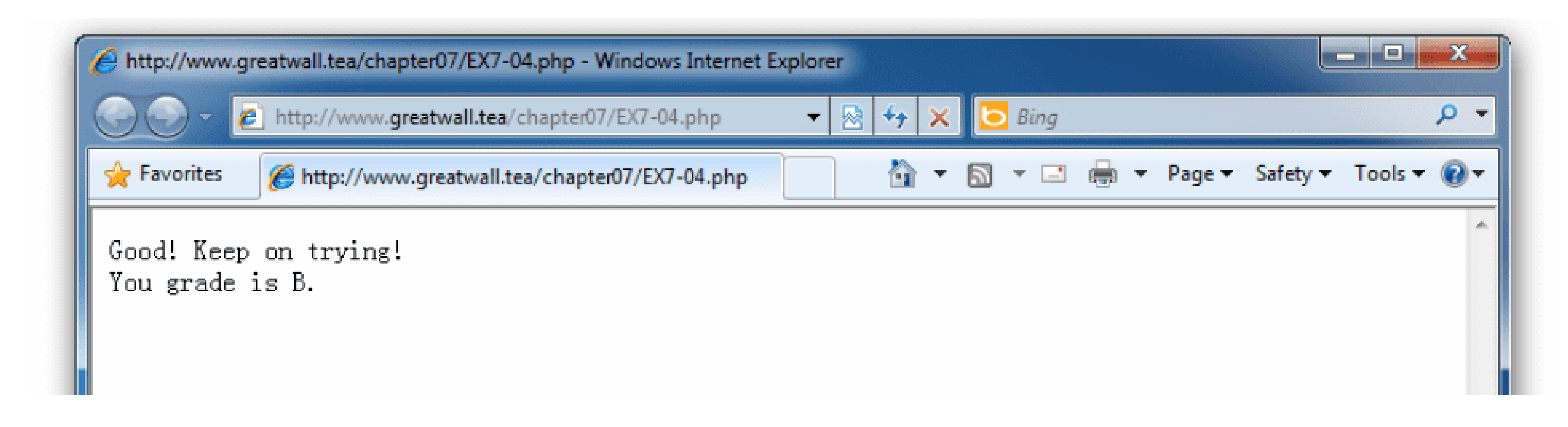


图 7-2　例 7.4 的执行结果

读者可以通过修改变量$actualScore 的值来改变脚本执行的结果。

有的时候，情况会变得有些复杂：当一个条件判断为真时，还需要再做出一个判断，才能得到想要的结果。比如例 7.5。

【例 7.5】 条件语句的嵌套的使用。

```
<?php
   if ($custProv == "HB") {
        if ($isEmailAdd == ""){
            $contactMethod = "letter";
        } else {
            $contactMethod = "Email";
        }
   } else {
        $contactMethod = "Not for this time";
   }
?>
```

这段脚本先对顾客所在的省份进行了判断：如果顾客不是来自湖北省，则忽略之（直接为变量$contactMethod 赋值“Not for this time”）。若顾客来自湖北省，则继续判断其是否登记了电邮地址。若其电邮地址为空，则向其发送函件（为变量$contactMethod 赋值“Letter”）；若登记了电邮地址，则向其发送电子邮件（为变量$contactMethod 赋值“Email”）。在这段脚本中，我们用了两个条件语句，其中一个嵌套在了另一个中间。这就叫条件语句的嵌套。

7.1.4　复杂条件语句 switch

在简单条件语句中，我们遇到的始终都是二选一的情况，非黑即白。然而在现实生活中，我们经常会面临多选一的抉择。用 PHP 脚本来描述这种多选一的情况，就得用到 switch 语句。

它的结构如下：

```
switch ($varName)
{
    case $value1 :
        statement block A;
        break;
    case $value2 :
        statement block B;
        break;
    case $value3 :
        statement block C;
        break;
    …
    default:
        statement block by default;
        break;
}
```

在这一段脚本中，PHP 处理引擎会测试变量$varName 的值，然后根据$varName 的值跳到相应的 case 部分执行相应代码块中的代码直到遇到 break 语句或 switch 语句末端。若被测试变量的值没有匹配任何 case 部分，PHP 处理引擎会直接执行默认部分代码。一个 switch 语句中 case 部分的数量由你来定，而默认部分的则是可选的。若在 switch 语句中添

加了默认部分，则最好将其放在所有 case 部分的后面。

下面这段脚本针对不同的省份，设置了不同的简称。一起来看看它是怎么做到的。

【例 7.6】 为省份设置简称。

```
switch ($province)
{
    case "Beijing":
        $abbrValue = "BJ";
        break;
    case "Hebei":
        $abbrValue = "HEB";
        break;
    case "Henan":
        $abbrValue = "HEN";
        break;
    case "Hubei":
        $abbrValue = "HB";
        break;
    default:
        $abbrValue = "Undefined";
        break;
}
```

在例 7.6 中，使用了 switch 语句来测试变量$province 的值。并提供了五种可能的取值：

- 若变量$province 的取值为“Beijing”，则变量$abbrValue 的取值为“BJ”。
- 若变量$province 的取值为“Hebei”，则变量$abbrValue 的取值为“HEB”。
- 若变量$province 的取值为“Henan”，则变量$abbrValue 的取值为“HEN”。
- 若变量$province 的取值为“Hubei”，则变量$abbrValue 的取值为“HB”。
- 若变量$province 的值未匹配上述任一取值，则变量$abbrValue 的取值为“Undefined”。

在使用 switch 语句时，需要特别注意，在结束每个 case 部分时，一定要使用 break 语句，告诉 PHP 处理引擎 switch 语句中余下的内容不用处理了，否则 PHP 处理引擎会顺序执行 switch 语句中余下的部分。我们还是以例 7.6 为例，若把其中的 break 语句全部去掉，无论变量$province 的取值是什么，变量$abbrValue 的取值均为“Undefined”。

7.1.5　实战练习：用户信息验证

在之前几个章节的实战练习中，我们都反复用到条件来控制脚本的运行。其实，对于一个稍微有些用处的 PHP 脚本来说，使用几个条件判断和循环往复来控制脚本的运行，是一件稀松平常的事情。在这一节里，我们将巩固本章里学到的内容来完成一项任务，那就是验证用户在表单中输入的各项信息是否符合要求。

看到这里，估计有的同学已经明白是怎么一回事儿了。是的，我们要使用正则表达式对用户输入的各项信息进行检查，以防止不符合要求的数据影响系统的稳定性。

在用户填写完成图 7-3 左侧的表单，并单击“提交”按钮后，在页面右侧会出现相应的信息。如果用户有未填写的信息，或者填写的信息不符合格式要求，均会出现相应的提示。只有当用户填写的信息完全符合格式要求时，才会出现如图 7-3 所示的内容。

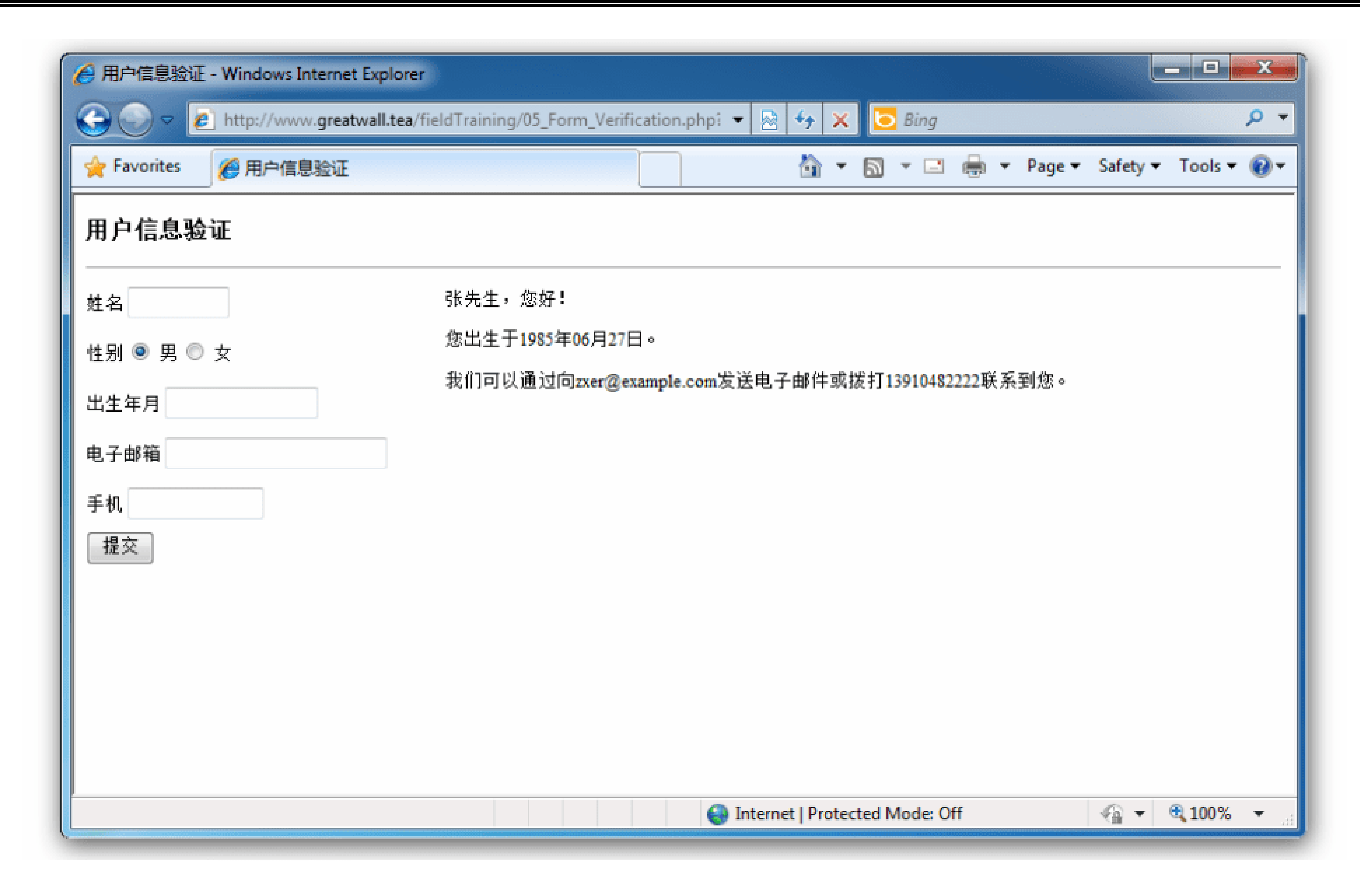


图 7-3　用户信息验证

我们来看看具体的实现过程。

（1）HTML 部分。

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset='UTF-8'>
        <title>用户信息验证</title>
        <style>
            body {
                font-size:small;
            }

            form div {
                line-height: 35px;
            }

            .form-container {
                width:30%;
                float:left;
            }

            .info-cotainer {
                width:80%;
                float:left;
            }

        </style>
    </head>
    <body>
        <h3>用户信息验证</h3>
        <hr />
        <div class="form-container">
            <form action="?check" method="post">
                <div>
                    姓名 <input type="text" size="6" name="user">
```

```
                </div>
                <div>
                    性别
                    <input type="radio" name="gender" value="male" checked> 男
                    <input type="radio" name="gender" value="female"> 女
                </div>
                <div>
                    出生年月
                    <input type="text" size="12" name="birthday">
                </div>
                <div>
                    电子邮箱
                    <input type="text" name="email">
                </div>
                <div>
                    手机
                    <input type="text" size="10" name="phone">
                </div>

                <button type="submit">提交</button>

            </form>
        </div>
        <div class="info-container" id="info"></div>   //信息会输出到这里
    </body>
</html>
```

在这个页面里，我们使用 CSS+DIV 布局把页面分成了三个部分，一个用于存放“用户信息验证”标题、一个用于位于标题下方左侧的表单、另一个则用于存放标题下方右侧的信息展示区域。

我们需要实现的功能是，验证用户在标题上方左侧的表单中输入的数据是否符合要求。若符合要求，则在标题下方右侧的信息展示区域中输出相应的信息，若不符合要求，则在标题下方右侧的信息展示区域中输出错误提示信息，提醒用户修改表单中的数据。

（2）PHP 部分。

```
<?php
    if(isset($_REQUEST['check'])){      //判断用户是否提交了表单
        $name = $_REQUEST['user'];
        $gender = $_REQUEST['gender'];
        $birthday = $_REQUEST['birthday'];
        $email = $_REQUEST['email'];
        $phone = $_REQUEST['phone'];    //获取用户输入的信息

        $msg = '';                      //定义输出消息的初始值

        if($name == '' ||
          $gender == '' ||
          $birthday == '' ||
          $email == '' ||
          $phone == ''){                //判断是否有未输入的表单项
            $msg = '<li>您至少有一项没有填写。</li>';
        } elseif (!preg_match('/^[0-9]{4}[-\/.][0-1][0-9][-\/.][0-3][0-9]$/',
        $birthday)){
            $msg = '<li>您输入的出生日期格式不正确。可接受的格式有:
            <ol><li>2013-07-06</li><li>2013/07/06</li><li>2013.07.06</li>
            </ol></li>';
```

```
/* 上面的正则表达式是判断用户输入的是否为可接受的三种格式之一。 */
} elseif (!preg_match('/^.+@.+\.com$/',$email)){
    $msg ='<li>您输入的邮件地址格式不正确。可接受的格式为：you@example.com。
    </li>';
/* 上面的正则表达式是判断用户输入的邮件地址格式是否正确。*/
} elseif (!preg_match('/^1[3458][0-9]{9}$/',$phone)){
    $msg = '<li>您输入的手机号码格式不正确。现只接受中国大陆地区的手机号码。
    </li>';
/* 上面的正则表达式是判断用户输入的手机号码是否为 13、14、15、18 开头的中国大陆
   地区的手机号码 */
} else {
    $title = mb_substr($name,0,1,'utf-8');
    if($gender == 'male') {
        $title .= '先生';
    } else {
        $title .= '女士';
    }                               // 获取用户的称谓

    $msg = '<p>'.$title.'，您好！</p>';
    $msg .= '<p>您出生于'.substr($birthday,0,4).'年'.substr
    ($birthday,5,2).'月'.
         substr($birthday,8,2).'日。</p>';
    $msg .= '<p>我们可以通过向'.$email.'发送电子邮件或拨打'.$phone.'联
    系到您。</p>';
}

echo '<script language="javascript">document.getElementById
("info").innerHTML="<ul>'.
$msg.'</ul>"</script>';          // 输出用户的信息到页面中指定的位置
    }
?>
```

在上面这段 PHP 脚本中，依然通过“?check”这样一个 URL 后缀来判断用户是否提交了表单（关于如何在页面间传递数据，请参考本书 9.1 节的内容）。

若用户提交了表单，我们将使用$_REQUEST 数组把用户输入的内容存放在$name、$gender、$birthday、$email 和$phone 五个变量中。然后再定义一个空字符串变量$msg 用于存放在对以上五个变量验证后产生的信息。若用户输入的信息符合要求，则将用户输入的信息重新整合后存入变量$msg 中；若用户输入的信息不符合要求，则将提示用户修改数据的信息存入变量$msg 中。

按照上述要求，我们在脚本中使用了“if…elseif…else…”结构和正则表达式实现了对用户输入数据的验证工作。

在这里，我们来详细了解一下脚本中使用的正则表达式：

- ^[0-9]{4}[-\/.][0-1][0-9][-\/.][0-3][0-9]$

这一条正则表达式用通俗的话来说，就是一个用来判断年月日格式的正则表达式。其中，年份必须为四位，月份和日期必须为两位。年、月、日之间由“-”、“/”或“.”分隔。

- ^.+@.+\.com$

这一条正则表达式则是用来判断电子邮件地址的，具体可以参考 7.1.1 小节中的内容。

- ^1[3458][0-9]{9}$

这一条正则表达式则规定了一串以“1”开头的 11 位数字串，并且第二位必须为 3、4、

5 或 8。

当用户输入的任何一条数据为空，或者无法匹配上述的正则表达式，则会导致系统在标题下方右侧输入对应的提示信息。而只有当用户输入了所有要求的信息，且生日、电子邮箱地址和手机号码均匹配对应的正则表达式，系统才会在标题下方右侧的页面中输出类似图 7-3 所示的信息。

7.2　重复性运算——循环

在这一章的开始，我们就说过：在实际生活中，除了需要不断地做出选择之外，我们还得不断重复着某些活动。无论你能否接受，C'est la vie。

为了模拟这个重复的过程，也为了让脚本变得更加轻巧，PHP 给出了解决重复性运算的方案，那就是循环。在 PHP 脚本中，可以使用三种类型的循环，如下所示。

- for 循环：在一个 for 循环中，需要设置一个计步器和一段需要重复执行的脚本。当计数器的值达到指定上限时，跳出循环。
- while 循环：在一个 while 循环中，需要设置一个条件供 PHP 处理引擎检测。若条件判断为 true，PHP 处理引擎重复执行某一段脚本直到条件判断为 false 时跳出循环。
- do…while 循环：在一个 do…while 循环中，将先执行一段脚本，然后设置一个条件供 PHP 处理引擎检测，若条件判断为 true，PHP 处理引擎重复执行之前执行的那一段脚本直到条件判断为 false 时跳出循环。

在本节里，我们就来看看在 PHP 脚本中，应该如何使用这些循环。

7.2.1　for 循环

上面说过，最基本的 for 循环需要定义一个循环变量（计步器），并为这个变量设置起始值、终结条件和步长。基本结构如下：

```
for (start value; end condition; increment)
{
    statement block
}
```

其中，三个变量之间是用两个分号（;）隔开的，第三个参数后面没有分号。

- *start value* 定义了循环变量的起始值。如果在这参数中写上“$i = 1”，那么循环变量就是$i，这个变量的起始值为 1。通常情况下，我们的循环变量都是用 0 或 1 做为起始值的，但是也可以使用其他数字或变量来定义循环变量的起始值。
- *end condition* 定义了循环变量的终结值。如果在这个参数中写上“$i<=10”，那么循环变量的终点为 10。只要终结条件为 true，循环就一直持续下去。只有当终结条件为 false 时，循环才会终结。循环变量的终结值可以是数，也可以是一个变量。
- *increment* 定义了循环变量的步长，也就是如何从起始值一步一步走到终结值的。如果在这个参数中写上“i++”或者“i=+1”，那么这个循环的步长为 1，也就是

说每循环一次，循环变量加 1。通常情况下，我们用设置循环的步长为 1，不过也可以使用其他的步长。

- *statement block* 定义了需要重复执行的语句。也就是说循环变量的值每改变一次，该语句块中的语句都会被执行一次。如果我们在这里写上“echo "Hello World!";”，那么只要终结条件为 true，PHP 处理引擎就会不断的输出“Hello World!”这句话。

我们把上面四段文字的解析写在一起就是下面这个样子。

【例 7.7】 一个简单的循环。

```
for ($i = 1; $i <= 10; $i++)
{
    echo "$i. ";
echo "Hello World!";
    echo "<br>";
}
```

这段脚本会以一行一句的形式输出 10 句“Hello World!”，如图 7-4 所示。

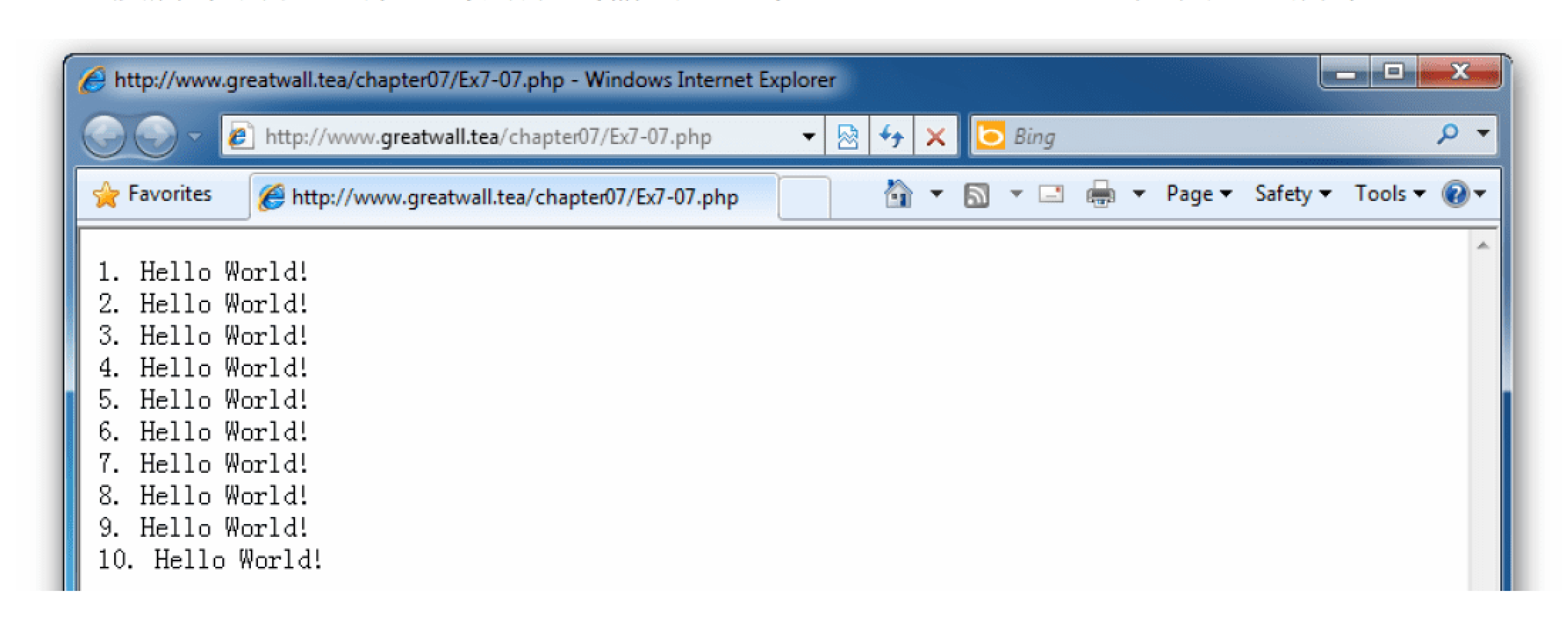


图 7-4　循环输出 Hello World!

for 循环还支持嵌套，也就是说我们可以在一个循环中进行另一个循环。比如，如果要输出九九乘法表的话，我们可以这么写。

【例 7.8】 九九乘法表。

```
<?php
    echo "<table border='0' cellspacing='5' cellpadding='5'>";
    for($x = 1; $x < 10; $x++){
        echo "<tr>";
        for($y=1; $y < 10; $y++){
            if($x >= $y){
                echo "<td>$y x $x = ".$x*$y."</td>";
            } else {
                echo "<td></td>";
            }
        }
        echo "</tr>";
    }
    echo "</table>";
?>
```

输出的结果如图 7-5 所示。

在这段脚本中，使用了 for 循环的嵌套。在第一个 for 循环中，我们定义了变量$x，起

始值为 1，步长为 1，终结值为 9。在第二个 for 循环中，我们定义了变量$y，起始值、步长和终结值与变量$x 相同。

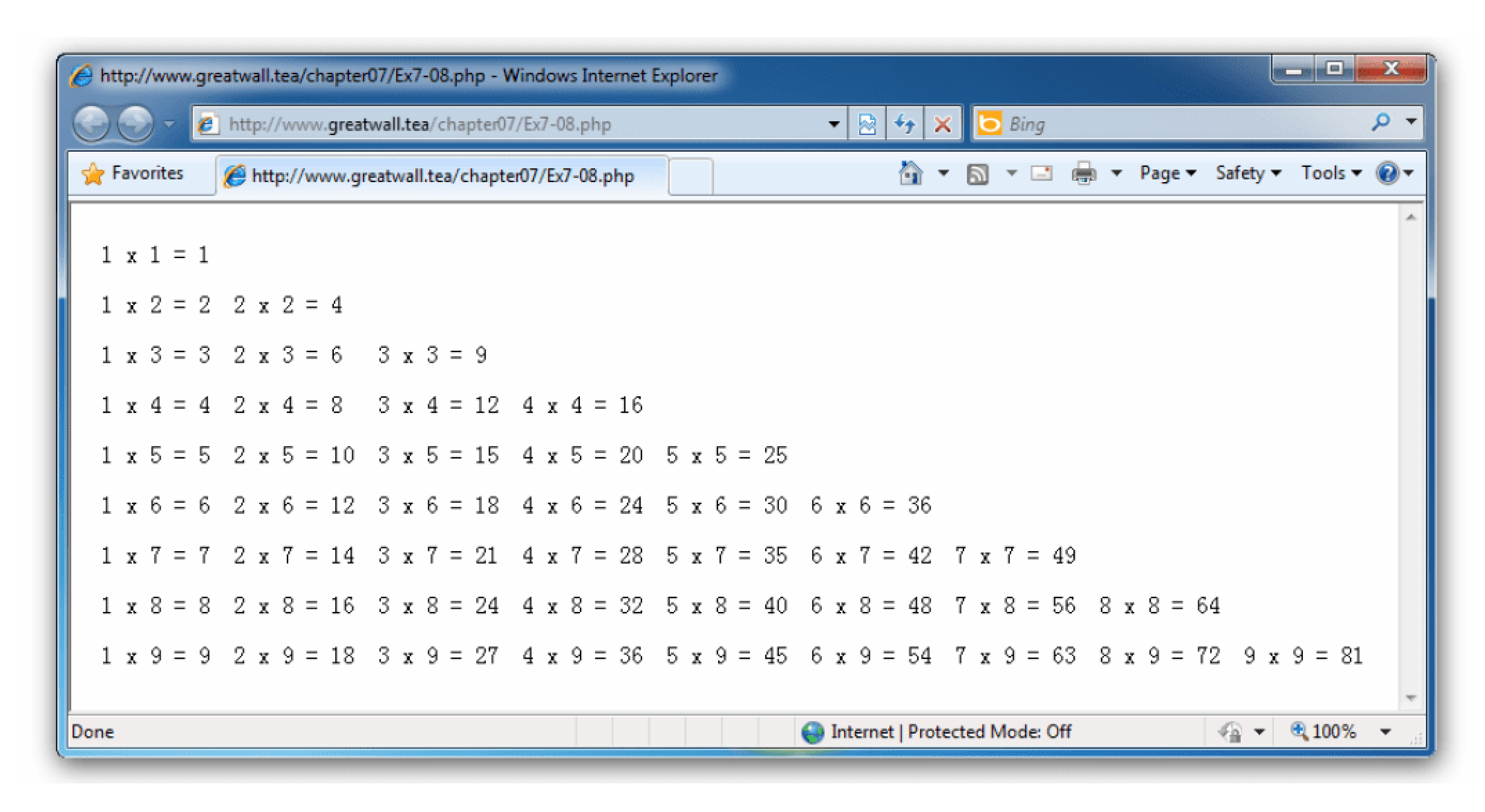


图 7-5　九九乘法表

在这段脚本中，外层循环只会被执行 9 次，而内层循环会在外层循环每执行一次时就执行 9 次，共计 81 次。我们一起来看看这段程序是按照什么顺序执行的：

PHP 处理引擎首先执行外层的 for 循环，获取变量$x 的值。继续往下执行。当碰到内层的 for 循环时，获取变量$y 的值。在内层的 for 循环中，我们还使用了一个 if 条件句，用来比较变量$x 和变量$y 之间的大小。当变量$x 的值大于等于变量$y 的值时，打印变量$x 和$y 的乘积。当变量$x 的值小于变量$y 的值时，打印空单元格。

在这例 7.8 中：

- ❑ 当变量$x 和变量$y 均为 1 时，打印“1 x 1 = 1”；
- ❑ 当变量$x 为 1，变量$y 为 2 时，打印空单元格；
- ❑ 当变量$x 为 2，变量$y 为 1 时，打印“1 x 2 = 2”；
- ❑ 当变量$x 为 2，变量$y 也为 2 时，打印“2 x 2 = 4”；
- ❑ ……

以此类推，得到如图 7-5 所示的结果。

其实上面的例子对于 PHP 来说只是小儿科。更为复杂的循环应该是下面这个样子的：

```
for(initial statement;
loop condition;
end-loop statement)
{
        statement blocks
}
```

其中，

- ❑ *initial statement* 定义了循环起始状态。这一部分的语句只在循环开始时执行一次。
- ❑ *loop condition* 定义了循环继续的条件。若条件判断为 false，则循环终止。
- ❑ *end-loop condition* 定义了完成每一次循环后需要执行的语句。

上面三项之间用两个分号（；）隔开，第三项后面没有分号。

再来看一个例子。

【例 7.9】 复杂循环与简单循环。

```
1   <table border="0" cellspacing="5" cellpadding="5">
2       <tr>
3           <td>Even Numbers</td>
4           <?php
5           //打印 10 以内的偶数
6               $t = 0;
7               for($i = 1, $j = 1;
8                   $t <= 8;
9                   $i++,$j++
10              ){
11              $t = $i + $j;
12              echo "<td>$t</td>";
13          }
14          ?>
15      </tr>
16      <tr>
17          <td>Even Numbers</td>
18          <?php
19              //打印 10 以内的偶数
20              for($i = 0; $i <= 10; $i+=2){
21                  if ($i > 0)
22                      echo "<td>$i</td>";
23              }
24          ?>
25  </tr>
26  </table>
```

在例 7.9 这段脚本中，使用了两个独立的 for 循环用来打印 10 以内的奇偶数。第一段 for 循环从第 7 行始到第 13 行止，是一个复杂循环。第二段 for 循环从 20 行始到第 23 行止，是一个简单循环。分别来看看这两段脚本：

在第一段的脚本中，我们在循环初始定义了两个变量，它们分别是变量$i 和变量$j。这两个变量的值均为 1。然后将变量$t 小于等于 8 作为循环继续的条件，变量$t 的原始值为 0。接着我们让变量$i 和变量$j 的值在每次循环后分别加 1。

在每次循环中，都将变量$i 和变量$j 的和赋值给变量$t，然后打印变量$t 的值到指定单元格。大家在这里可能会有一个疑问：为什么这一段脚本最后会输出 10 这个偶数？循环难道不应该在变量$t 大于 8 的时候就终止了吗？

带着这个疑问，我们再来看看这一段脚本。假设此时变量$i 和变量$j 均为 4，与此同时变量$t 的值应该为 6 而不是 8。因为在上一轮的循环结束前，也就是当变量$i 和变量$j 均为 3 时，PHP 处理引擎将变量$i 和变量$j 的和赋值给了变量$t。因此当变量$i 和变量$j 为 4 时，变量$t 的值为 6。在这一轮循环结束后，变量$t 的值变成了 8，而变量$i 和变量$j 的值则变成了 5。由于此时变量$t 的值仍然是小于或等于 8 的，于是循环继续，系统输出变量$i 和变量$j 的和，也就是 10，然后变量$i 和变量$j 各自加 1，回到循环起点。这时变量$t 的值已经为 10，循环继续的条件不成立，循环结束。

在第二段脚本中，我们在循环初始定义了变量$i，并为其赋值 0。然后将变量$i 小于等于 10 作为循环继续的条件。最后，我们让变量$i 在每次循环结束后加 2。在每次循环中，只要变量$i 的值大于 0 都会被输出。于是就看到了一串小于 10 的偶数。

例 7.9 所示中的两段脚本殊途同归，分别展现了复杂循环和简单循环的不同魅力。输出结果如图 7-6 所示。

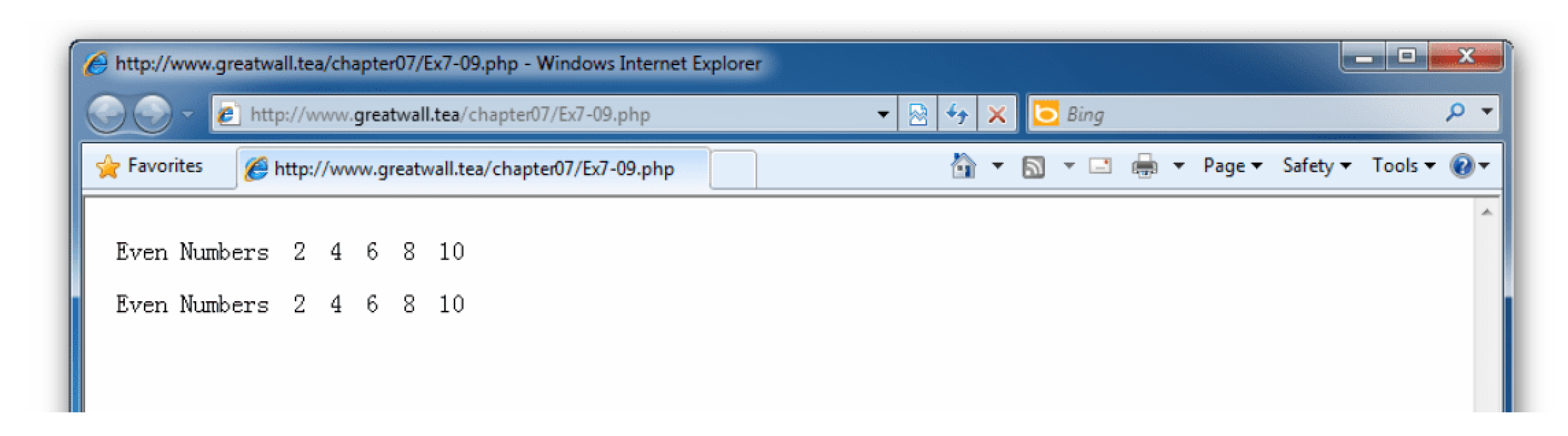


图 7-6　复杂循环与简单循环

7.2.2　while 循环

在 7.2 节的开始，我们说过：“在一个 while 循环中，需要设置一个条件供 PHP 处理引擎检测。若条件判断为 true，PHP 处理引擎重复执行某一段脚本直到条件判断为 false 时跳出循环。”具体来说一个 while 循环有两个部分：

- ❑ 一个条件供 PHP 判断，
- ❑ 一段需要循环执行的脚本。

当条件判断为 true 时，循环继续；当条件判断为 false 时，循环结束。

如此看来，while 循环的结构应该是下面这个样子：

```
while(condition)
{
    statement block
}
```

下面，用 while 循环模拟一下我们在一堆铜钮扣中找一颗金钮扣的过程吧。

【例 7.10】 寻找金钮扣。

```
<?php
    //准备一些扣子
    $buttons = array("copperButton1",
                     "copperButton2",
                     "copperButton3",
                     "copperButton4",
                     "copperButton5",
                     "goldButton",
                     "copperButton6");

    //准备一些在找不着金钮扣时需要说的话
    $complaints = array("Ooops! Not the gold one!<br>...<br>",
                        "Uh oh! What a mess! where is the gold button?
                        <br>...<br>",
                        "Sigh! Still not the gold button! Where did I put
                        it?<br>...<br>",
                        "Ahhha! This is it!!!! Uh, it seems a little bit
                        different. Sigh!<br>...<br>");

```

```
17      //准备一些在寻找金钮扣过程中需要的变量
18      $testVar = "no";
19      $counter = 0;
20
21      //开始找金钮扣
22      while ($testVar != "yes"){
23          $complaintRandom = mt_rand(0, 3);
24          if ($buttons[$counter] == "goldButton"){
25              $testVar = "yes";
26              echo "Hooray! The gold button is now mine!<br>";
27          } else {
28              echo $complaints[$complaintRandom];
29          }
30          $counter++;
31      }
32  ?>
```

在例 7.10 这段脚本中，定义了一堆钮扣（数组$buttons），其中只有一颗是金钮扣（goldbutton）。然后又定义了一系列抱怨没有找到钮扣时可以说的话（数组$complaints）。紧接着定义了用于测试当前对象是否为金钮扣的变量$testVar 和计步用的变量$counter。如果当前找到的钮扣不是金钮扣，那么变量$testVar 的值不变。如果当前找到的钮扣是金钮扣，那么变量$testVar 的值变成“yes”。变量$counter 的值初始为 0，每次循环结束后加 1。

从第 21 行开始，我们使用了一个 while 循环，用于模拟我们找钮扣的过程。这个循环得以继续的条件是变量$testVar 的值不为“yes”。由于循环开始时变量$testVar 的值不为“yes”，PHP 开始执行 while 循环内部的语句。

在 while 循环内部使用了一个 if 语句用来判断我们找到的是不是金钮扣：如果是，则将变量$testVar 的值改为“yes”并打印“Hooray! The gold button is now mine!
”；如果不是，则随机输出一句抱怨的话。

若在某次循环的过程中，变量$testVar 的值变成了“yes”，那么 PHP 在下一次循环开始前就会发现循环继续的条件不成立，循环结束。

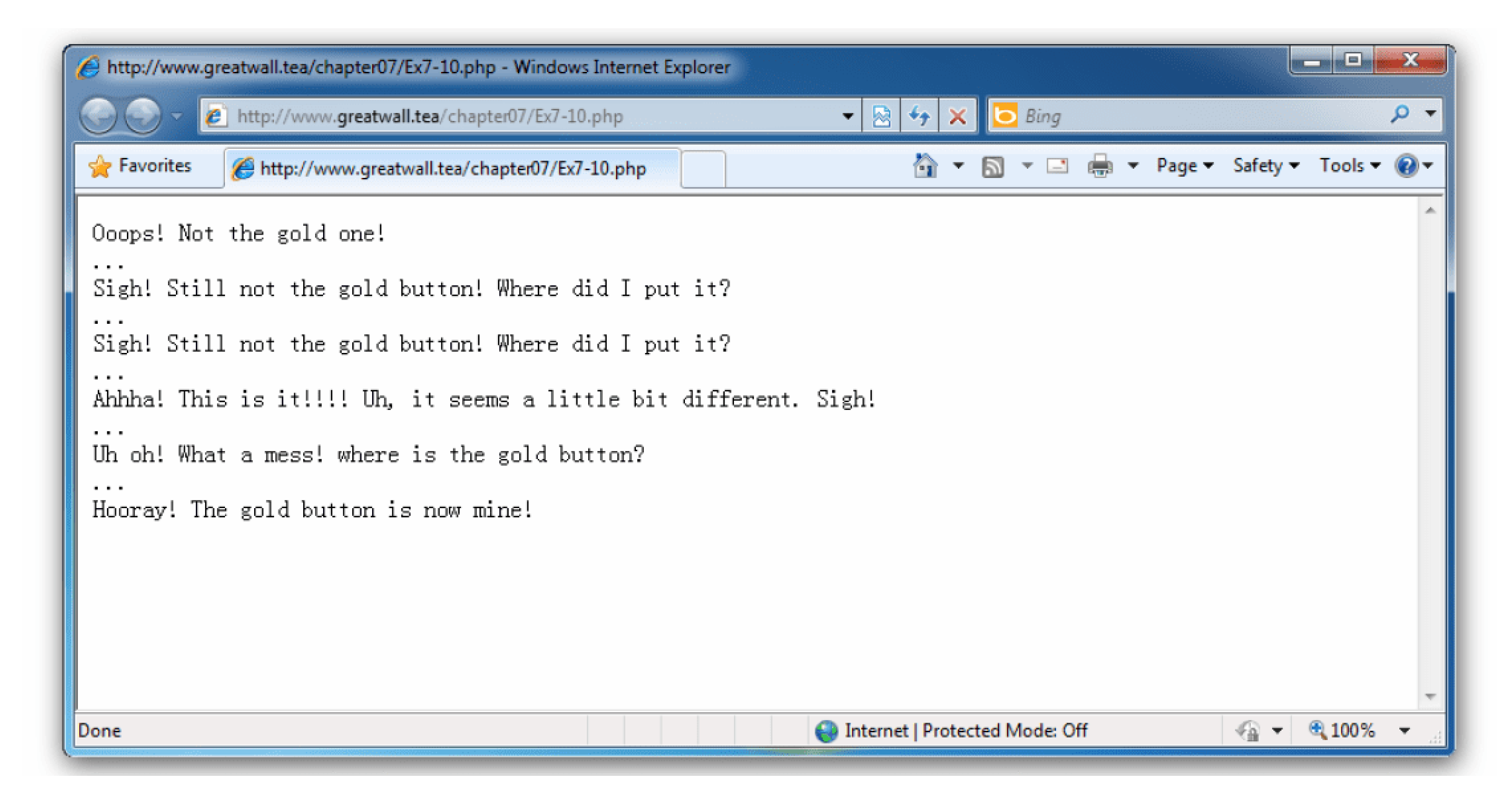


图 7-7　寻找金钮扣

图 7-7 展示了该脚本执行的结果。需要注意的是，由于没有找到金钮扣时的抱怨是随机产生的，所以每次输出的语句可能不一样。

实话说，这段脚本模拟寻找金钮扣的过程有些不太真实。虽然每次执行过程中，输出的抱怨都不一样，但是金钮扣都是在五次失败后找到的。如果想要找到金钮扣的时间点更加随机，应该怎么办呢？

解决这个问题关键点就在变量$counter 上。变量$counter 的值是随着循环次数的增加逐渐由小变大的。由于变量$counter 在循环中充当数组$buttons 的索引，而金钮扣是数组$buttons 中的第六个元素，所以找到金钮扣的时间点就固定了。为了让这个时间点更加随机，我们只需要让变量$counter 随机取值即可。

找到了思路，再回过头去看看例 7.10，发现只需要把第 19 行修改为下面的样子：

```
19  $counter = mt_rand(0,4);
```

修改好后，不断地执行这个脚本，会发现脚本执行的结果每次都不同。不光是抱怨的句子不一样，连找到金钮扣的时间点都变得随机起来了。

7.2.3　do … while 循环

do…while 循环和 while 循环类似，它们都是通过判断一个条件来决定循环是否继续的。但是 do…while 循环和 while 循环不一样的地方，就是 do…while 会在判断一个条件是否成立之前就预先执行一次需要循环往复执行的脚本。它的结构应该是下面这个样子：

```
do
{
    statement block
} while (condition);
```

特别要留心 condition 后面的分号，这个是不能丢掉的。

我们来看一段脚本，体现一下 do…while 循环和 while 循环的区别。

【例 7.11】 do…while 循环和 while 循环的区别。

```
<?php
    //计步器初始化
    $counter = 0;

    //使用一个 while 循环
    while ($counter == 1){
        $counter++;
    }
    echo "The first loop is executed for $counter times.<br>";

    //计步器重新初始化
    $counter = 0;

    //使用一个 do...while 循环
    do
    {
        $counter++;
    } while ($counter == 1);
    echo "The second loop is executed for $counter times.";
?>
```

在例 7.11 这段脚本中，我们分别使用了一个 while 循环和一个 do…while 循环。在这两个循环中，我们均使用了变量$counter，用来记录需要循环执行的脚本被执行的次数。在第一个脚本中，我们使用的是 while 循环，在该循环中，循环可继续的条件是变量$counter 的值为 1，在每次循环时变量$counter 的值加 1；在第二个脚本中，我们使用的是 do…while 循环，在该循环时，循环可继续的条件依然是变量$counter 的值为 1，在每次循环时变量$counter 的值加 1。

现在来看看执行结果，如图 7-8 所示。

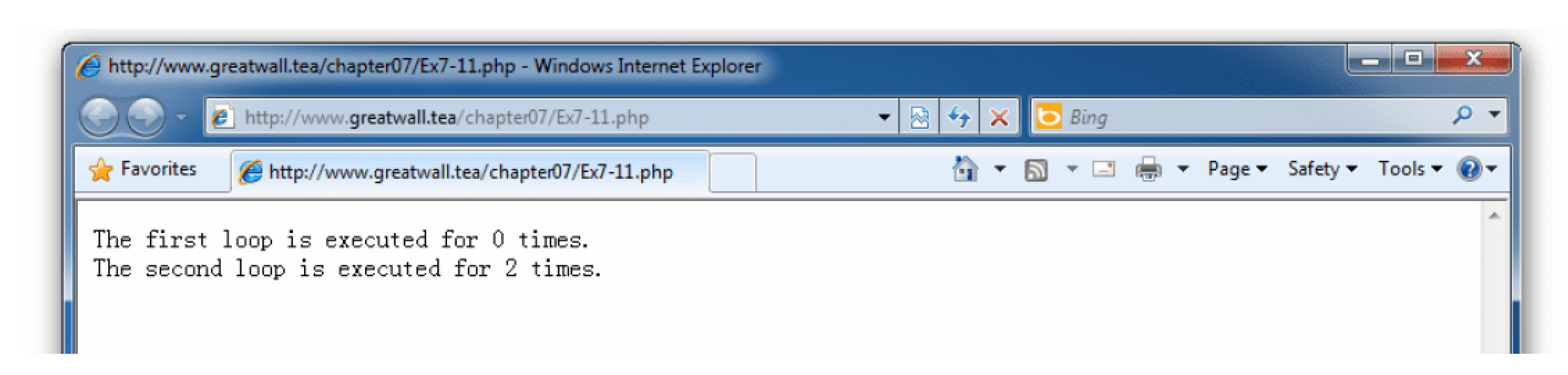


图 7-8　do…while 循环和 while 循环的区别

我们惊奇地发现，第一段脚本执行的次数为 0，而第二段脚本执行的次数为 2。这是为什么呢？

先来看第一段脚本，变量$counter 的初始值为 0。这样一来，循环继续的条件从一开始就不成立。因此变量$counter 的值不会发生变化，需要循环执行的脚本一次也没有执行。

而在第二段脚本中，虽然变量$counter 的初始值也为 0，但在没有进行条件判断之前就已经加了 1。这样一来，变量$counter 的值就变成了 1。再进行条件判断时，循环继续的条件成立，于是变量$counter 的值又可以加 1，最后它的值就变成了 2。然后再进行条件判断，这时条件不成立，循环终止。这样一来，需要循环执行的脚本被执行的次数就成了两次了。

如果把循环是否可以继续的条件改成如下几种，大家猜猜结果会是怎样的呢？

- $counter < 1；
- $counter == 2；
- $counter <= 1。

这里只告诉大家，使用第一种和第三种条件得到的结果是一样的，使用第二种条件得到的结果与例 7.11 类似，但也许还是会出乎你的意料。具体原因大家自己分析一下吧。

7.2.4　避免无限循环

看到这儿，有的同学可能就不淡定了。为什么会无限循环呢，不是都设定条件了吗？只要条件不成立，循环就自己结束了。哪儿来的无限循环呢？我们来看下面一段脚本。

【例 7.12】 无限循环。

```
<?php
    $fruit = array("orange", "apple", "pear");
     $testVar = "no";

     while (!$testVar == "yes"){
```

```
        $key = 0;
        if ($fruit[$key] == "apple"){
            $testVar = "yes";
            echo "apple";
        } else {
            echo "This $fruit[$key] is not an apple.<br>";
        }
        $key++;
    }
?>
```

这段脚本无论在哪个 IDE 里都是不会报错的。在浏览器中也可以运行，但是却看不到结果。因为这段脚本会在后台不停地输出一句话，那就是“The orange is not an apple.”。为什么呢？

因为我们把给变量$key 赋值的那一句条件放在循环里面了。这样一来，每次循环开始时，变量$key 的值就被重新设置为 0 了。因此我们永远也等不到循环条件不成立的那一天，除非手动强行终止 PHP 进程。因此大家在写脚本的时候，务必要小心在循环内部给变量赋值。一招不慎就有可能出现无限循环的情况。

7.2.5　实战练习：遍历数组的另类方法

对于一个数组而言，用 foreach 对它进行遍历，不仅可以获取各数组元素的值，同时也可以获取各元素的值对应的键名。其实，我们也可以使用 for 循环和 while 循环来遍历一个数组。在本小节里，我们就来看看如何使用 for 循环和 while 循环来分别遍历一个数组并输出其中的键值对吧。

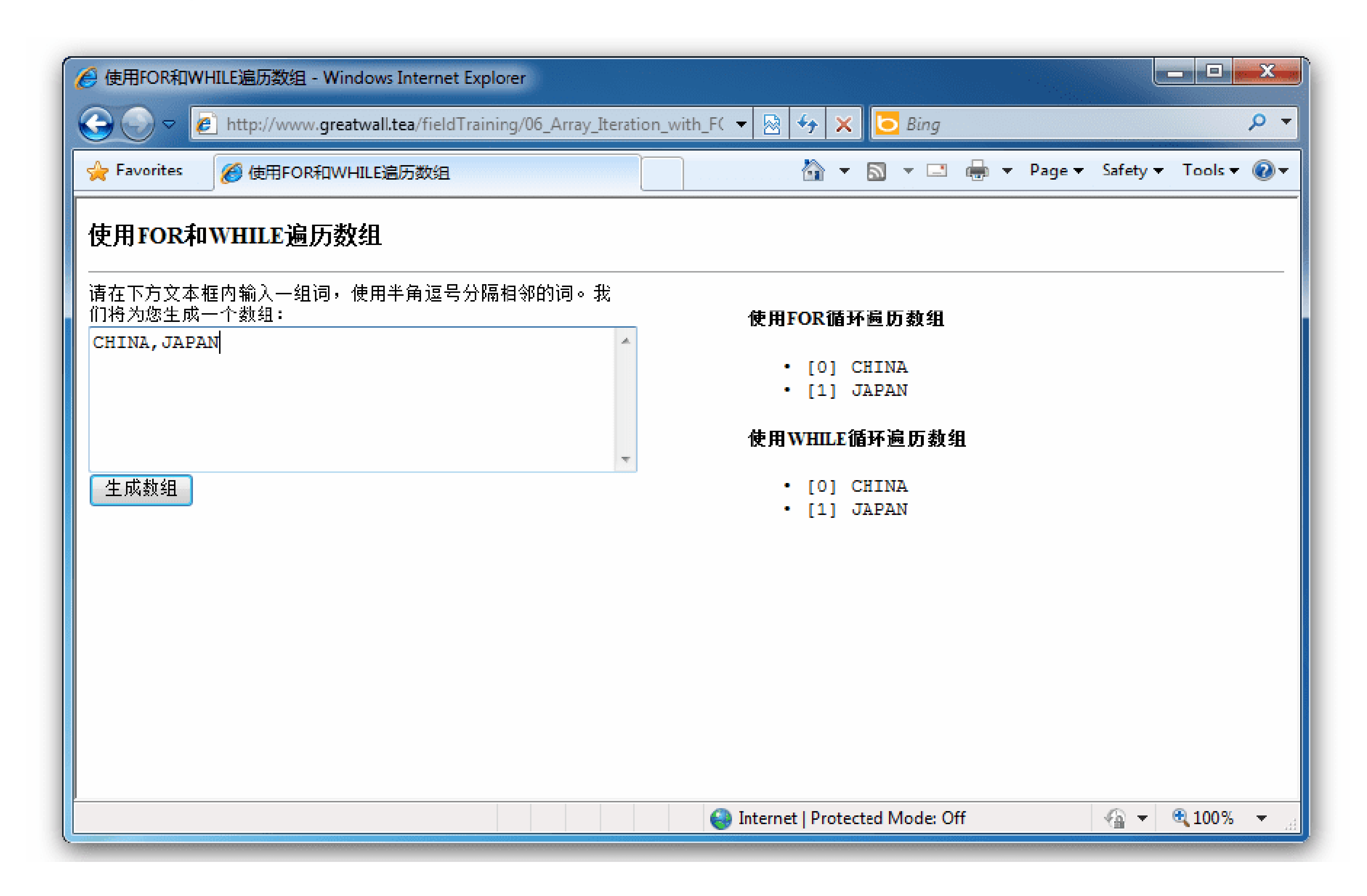


图 7-9　使用 FOR 和 WHILE 循环遍历数组

从图 7-9 中可以看到，使用 for 和 while 循环输出的数组结构是一模一样的。但是实现的方法就大相径庭了。我们先一起来看看这个脚本的 HTML 部分：

```
<!DOCTYPE html>
```

```
<html>
    <head>
        <meta charset="UTF-8">
        <title>使用 FOR 和 WHILE 遍历数组</title>
        <style>
            body {
                font-size:small;
            }

            .form-container{
                width:45%;
                float:left;
                margin-right:5%;
            }

            .info-container{
                width:45%;
                float:left;
                margin-left:5%;
            }

            li {
                font-family:monospace;
            }

        </style>
    </head>
    <body>
        <h3>使用 FOR 和 WHILE 遍历数组</h3>
        <hr />
            <div class="form-container">
                <form action="?check" method="post">
                    <div>
                        请在下方文本框内输入一组词，使用半角逗号分隔相邻的词。
                        <textarea rows="6" cols="45" name="arrayInput">
                        </textarea>
                    </div>

                    <button type="submit">生成数组</button>
                </form>
            </div>

            <div class="info-container" id="info">     /*--用于存放输出的内
                                                              容--*/

            </div>
    </body>
</html>
```

在这个页面中使用了与上个实战练习中相同的页面结构，即使用 CSS+DIV 将页面分成了三个部分：一个用来存入页面的标题“使用 FOR 和 WHILE 遍历数组”、一个用来存放表单、一个用来存放使用 for 循环和 while 循环输出的数组。

在页面中定义的表单，其“action”属性依然是“?check”，而其“method”属性依然是“post”。这将表明，下面使用 PHP 脚本，将会使用当前页面来处理用户提交的数据。

现在再来看看脚本的 PHP 部分：

```
<?php
    if(isset($_REQUEST['check'])){                         //检查用户是否已提交脚本
```

```
        $array = explode(',',$ REQUEST['arrayInput']); //获取用户输入的数组
        $msg = '<h4>使用 FOR 循环遍历数组</h4><ul>';
        if(sizeof($array) > 1) {                    //当数组元素的数量大于 1
                                                      时开始循环输出
            for ($i=0; $i < sizeof($array); $i++) {
                if($i == sizeof($array)-1) {        //当遍历到数组的最后一个
                                                      元素时在结尾加上"<ul>"
                    $msg .= '<li>['.key($array).'] '.current($array)
                    .'</li></ul>';
                } else {
                    $msg .= '<li>['.key($array).'] '.current($array).
                    '</li>';
                    next($array);                   //移动数组指针指向下一个元素
                }

            }

            reset($array);                          // 重置数组指针
            $msg .= '<h4>使用 WHILE 循环遍历数组</h4><ul>';
            $a = 0;                                 // 定义用于循环的计步器
            while ($a < sizeof($array)) {
                if($a == sizeof($array)-1){
                    $msg .= '<li>['.key($array).'] '.current($array)
                    .'</li></ul>';
                } else {
                    $msg .= '<li>['.key($array).'] '.current($array).
                    '</li>';
                    next($array);
                }
                $a++;
            }
        } else {
            $msg = "<li>请使用半角逗号分隔相邻的词</li></ul>";
            // 若用户输入的数组元素数量不足时提示用户
        }

        echo '<script language="javascript">document.getElementById
        ("info").innerHTML="'.$msg.'"</script>';
    }
?>
```

在脚本中，我们通过“isset($_REQUEST['check'])”来判断用户是否提交了表单。若用户提交了表单，则使用 explode()函数将用户输入的数据转换成一个数组存放在变量$array 中。然后使用 for 和 while 循环分别遍历数组$array 中的元素，并输出到变量$msg 中。

现在我们分别来看看使用 for 循环和 while 循环遍历数组元素的脚本。

（1）使用 for 循环遍历数组元素：

```
        $msg = '<h4>使用 FOR 循环遍历数组</h4><ul>';
        if(sizeof($array) > 1) {          //当数组元素的数量大于 1 时开始循环输出
            for ($i=0; $i < sizeof($array); $i++) {
                if($i == sizeof($array)-1) {    //当遍历到数组的最后一个元素
                                                  时在结尾加上"<ul>"
                    $msg .= '<li>['.key($array).'] '.current($array).
                    '</li></ul>';
                } else {
                    $msg .= '<li>['.key($array).'] '.current($array).
                    '</li>';
```

```
                    next($array);                  //移动数组指针指向下一个元素
                }

            }
```

在这段脚本中，我们将数组中的元素揉合到了一个无序列表中。若用户在页面左侧的文本框中输入的字符串中没有使用半角逗号的话，则“sizeof($array) > 1”的判断为 false，这时，页面标题下方右侧不会输出任何信息。只有当“sizeof($array) > 1”的判断为 true 时，也就是用户在页面左侧的文本框中输入的字符串中至少带有一个半角逗号时，输出数组元素的索引和值到变量$msg 中。

值得注意的是，由于在输出数组$array 的最后一个元素时，需要关闭无序列表，所以在 for 循环中加入了判断当前元素是否为数组$array 的最后一个元素的 if…else…语句。其实也可以不使用这个判断，而直接在遍历完数组$array 后，再关闭无序列表，如下所示：

```
        $msg = '<h4>使用 FOR 循环遍历数组</h4><ul>';
        if(sizeof($array) > 1) {                   //当数组元素的数量大于 1 时开
                                                     始循环输出
            for ($i=0; $i < sizeof($array); $i++) {
                $msg .= '<li>['.key($array).'] '.current($array).'</li>';
                next($array);                      //移动数组指针指向下一个元素
            }
            $msg .= '</ul>';
        }
```

（2）使用 while 循环遍历数组元素：

为了能够使用 while 循环对同一数组进行遍历，我们需要在使用 while 循环遍历数组前重置数组游标。因此，在使用 while 循环前使用了“reset($array)”函数。

在 while 循环内，遍历数组的思路与在 for 循环中类似，在这里就不再重复了。

只需要关注一点，那就是在糅合 PHP 变量与 HTML 代码时，记得多使用“.=”操作符，以缩短字符串表达式的长度，避免可能出现的书写错误。

7.3　习　　题

（1）请使用 if…else…语句结合 date()函数编写一段脚本，用来判断当前的季节（春、夏、秋和冬）。

（2）请使用 switch 语句结合 date()函数编写一段脚本，用来判断当前时段（早、中和晚）。

（3）请使用 for 循环输出 2013 年 10 月 15 日到 2013 年 11 月 15 日之间的所有日期。

（4）请使用 while 循环输出 1 到 10 之间（含 1 和 10）所有整数的和。

（5）请使用 do…while 循环输出 1 到 10 之间（含 1 和 10）所有整数的和。

第 8 章　脚本的重用

在本章里，我们将接触函数、类和对象这几个概念，它们都涉及到脚本的重用。那么什么是脚本的重用呢？通俗地说，脚本的重用就是把需要反复使用的脚本单独提取出来做成一个独立的函数或类，然后在需要使用到的地方使用这个函数或建立属于这个类的对象。

打个通俗的比方，交警同志们在指挥交通时做的那一套动作就可以被定义成一个类，我们可以叫它“交警类”。在这个类里，可以定义如下 8 个函数：

- “变道()”函数用来指挥车辆变换行驶道路；
- “减速慢行()”函数用来指挥车辆减速；
- “靠边停车()”函数用来指挥车辆靠边停车；
- “停车()”函数用来指挥车辆停车；
- “右转()”函数用来指挥车辆右转弯；
- “直行()”函数用来指挥车辆沿直线行驶；
- “左待转()”函数用来指挥车辆左转弯待转；
- “左转()”函数用来指挥车辆左转弯。

在定义完成这些函数后，我们就可以在需要使用相应动作的地方直接建立一个“交警类”的对象，然后通过这个对象使用这些函数。

8.1　自定义函数

在前面的章节里，我们见过了大大小小的函数也有不少了。虽然它们功能各不相同，不过都有一个共同的特点，那就是它们都是 PHP 的预置函数。使用预置函数的好处就在于，可以很方便地实现某些功能，而不用自己去写了。比如在使用取绝对值的“abs()”函数时，只需要指定一个取绝对值的数就成了，至于怎么计算已经在这个函数里写好了，我们就不必操心了。

这种可以简化脚本的方法，我们也可以借鉴一下。这就是自定义函数的源起。

8.1.1　小试牛刀

本小节将试着编写两个自定义函数，来体验一下自定义函数给我们的编码带来的便捷。

比如，我们经常会对某些文字进行加粗。在 HTML 里，最常见的给文字加粗的方法就是在需要加粗的文字两端加上“”标记对。为了避免重复添加这个标记对，可以编写一段脚本来简化给文字加粗的过程。

【例 8.1】 文字加粗函数的使用。

```
1    <?php
2        function bold($string){
3            echo "<b>".$string."</b>";
4        }
5
6        echo "This is not bold.<br>";
7        bold("This is bold.");
8        echo "<br>Again, this is not bold";
9    ?>
```

在这段脚本开始，我们就使用 function 关键字定义了一个名为 bold 的函数，并为这个 bold()函数指定了一个字符串类型的参数。这个函数内部需要重用的语句只有一句，作用是在变量$string 前后加上“”和“”标签，然后把这个连接起来的字符串打印出来。在接下来的第 6 行和第 8 行里，使用了 echo 关键字，打印不带文字加粗标记对的字符串；而在第 7 行里，使用了自定义的 bold()函数，并将变量$string 替换成了需要加粗的字符串。

这段脚本的运行结果如图 8-1 所示。

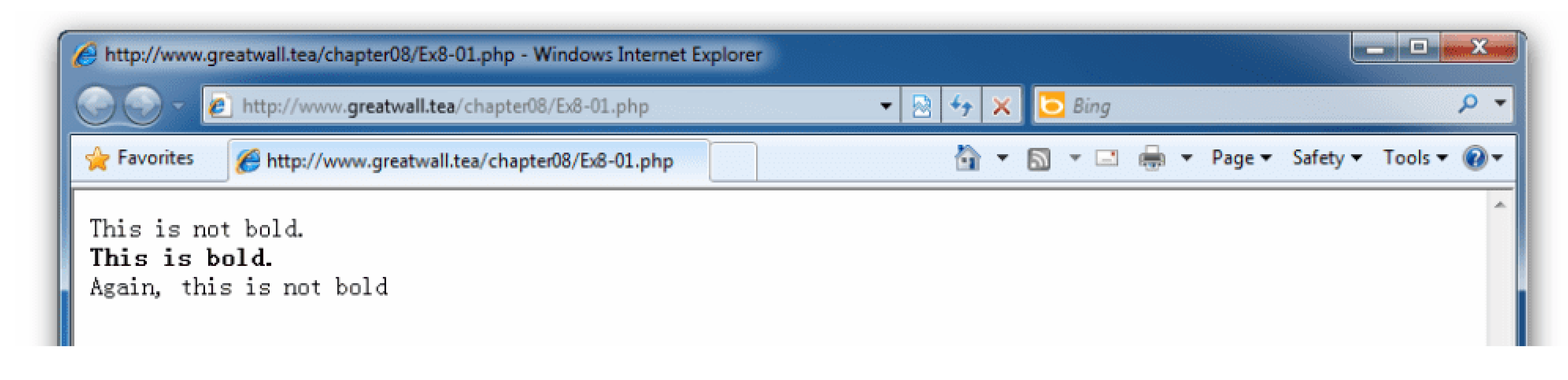


图 8-1　文字加粗函数

在显示的结果中，第二句话显然被加粗了，而第一句和第三句都没有。这样看来，bold()函数的编写是正确无误的，重用代码的目的也达到了。之后，我们可以在脚本的其他需要加粗文字的地方重复地使用这个函数。

现在来总结一下如何自定义一个函数。

定义一个自定义函数需要使用 function 关键字，然后指定自定义函数的名字和函数运算需要的变量，最后将需要重复使用的脚本包含在函数中。也就是下面这个样子：

```
function functionName(parameter1, parameter2, …)
{
    statement block
}
```

自定义函数除了像例 8.1 中的 bold()函数一样可以执行某个动作之外，还可以返回一个值供其他函数使用和处理。

在编写 HTML 文件时，某些标题需要使用一号标题样式（“<h1></h1>”），而有些标题需要使用二号标题样式（“<h2></h2>”）。重复的添加这些标签十分的繁琐。为此，我们可以编写一个 headings()函数，为不同层级的标题应用不同的样式。

【例 8.2】 标题样式函数的使用。

```
<?php
    function headings($string, $level){
        switch ($level) {
            case 1:
                $string = "<h1>$string</h1>";
                break;
            case 2:
```

```
                $string = "<h2>$string</h2>";
                break;
            case 3:
                $string = "<h3>$string</h3>";
                break;
            default:
                $string = "<p><b>$string</b></p>";
                break;
        }
        return($string);
    }

    $testString1 = headings("This is a second-level heading.", 2);
    $testString2 = headings("This is a string without any style.", 0);
    echo $testString1;
    echo $testString2;
?>
```

这段脚本定义的 headings()函数，一共带有两个参数：变量$string 和变量$level。在这个自定义函数里，使用了一个 switch 语句来检测变量$level 的值。若变量$level 的值为 1，则在变量$string 的前后加上“<h1>”和“</h1>”标签；若变量$level 的值为 2，则在变量$string 的前后加上“<h2>”和“</h2>”标签，依次类推。headings()函数只定义了三个层级的标题，因此我们在 switch 语句中使用了 default 关键字，用来应对当变量$level 的值不在 1 到 3 之间的情况。在函数的最后，使用了 return()函数返回变量$string 的值。

在函数外，定义了两个变量$testString1 和$testString2（假设变量$testString1 是个二级标题，而变量$testString2 是段普通文本），然后使用 headings()函数对它们赋值。最后使用 echo 关键字输出这两个变量。

这段脚本的运行结果如图 8-2 所示。

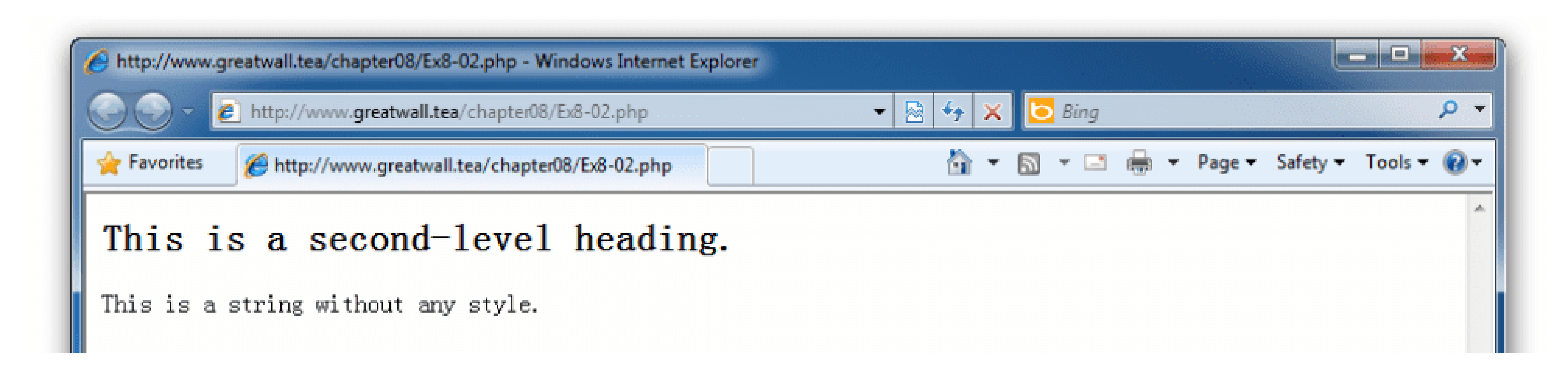


图 8-2　标题样式函数

8.1.2　参数与返回值

上一个小节定义了两个函数 bold()和 headings()，这两个自定义函数都带有参数，而且 headings()函数还返回了一个值供其他语句处理。在本小节里，我们就来仔细研究一下参数与返回值的变量类型。

现在先回过头去看看例 8.2，脚本里的两个参数$string 和$level 到底算是什么类型的变量呢？实际上，PHP 允许向变量传递任何类型的值，而且参数变量的类型是由传递给它的值决定的。在例 8.2 的脚本中，变量$string 应该是字符串类型的变量，而$level 应该是个整型变量。因为在函数定义的过程中，我们就是把$string 当成字符串，而把$level 当成整数在使用的。

再来看一个例子。

【例 8.3】 除法的使用。

```
<?php
    function divide($a,$b){
        return ($a/$b);
    }

     $c = divide (4, 2);
     var_dump($c);
     echo "<br>";
     $c = divide (3, 2);
     var_dump($c);
     echo "<br>";
     $c = divide (4.5, 2);
     var_dump($c);
?>
```

这段脚本定义了一个 divide()函数，它带有两个变量，分别是变量$a 和变量$b。这两个参数可以是任何类型的变量。在函数里，我们使用 return()函数返回了变量$a 除以变量$b 的值。在函数外，我们定义了变量$c，然后使用 divide()函数为其赋值，之后使用 var_dump()函数输出变量$c 的值和变量类型。

这段脚本运行的结果如图 8-3 所示。

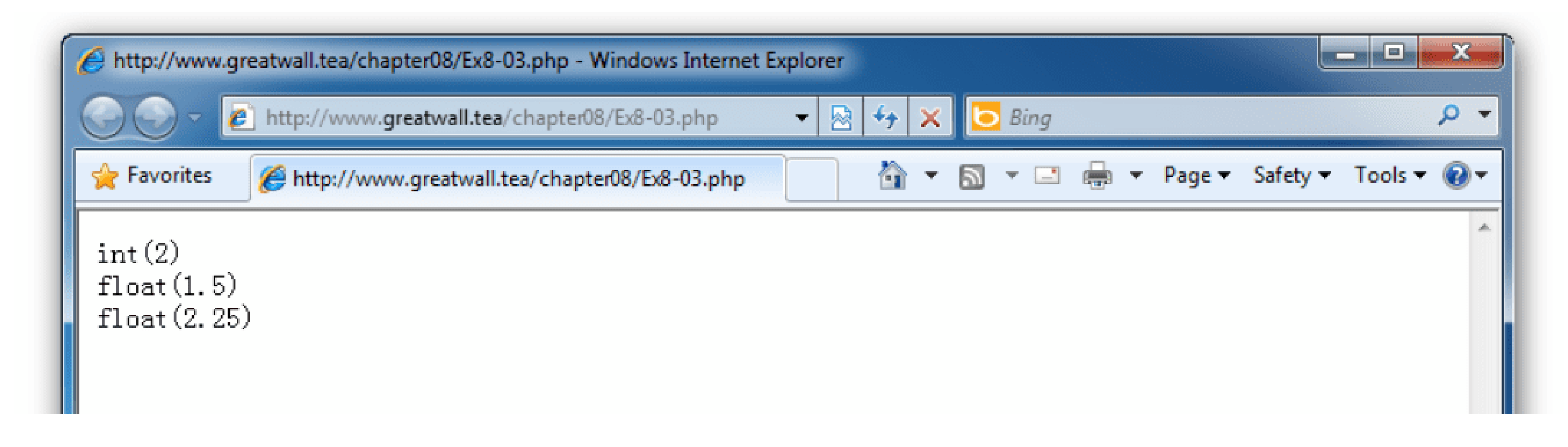


图 8-3　除法函数

在脚本中，我们三次使用 divide()函数为变量$c 赋值并输出其值和变量类型，发现三次结果有些差异：

- ❑ 第一次输出的是“4 除以 2”的值，为整型变量。
- ❑ 第二次输出的是“3 除以 2”的值，为浮点型变量。
- ❑ 第三次输出的是“4.5 除以 2”的值，为浮点型变量。

这样看来，变量的类型的确是由赋给它的值的类型来决定的。

我们知道除法中的被除数不应该为 0，否则除式将无法计算。但是定义的 divide()函数似乎没有采取任何可靠的措施来避免产生无法计算的情况。下面将 divide()函数修改成如下的样子，然后再把 divide(5,0)赋值给变量$c：

```
function divide($a, $b){
    if ($b == 0){
        return FALSE;
    } else {
        return ($a/$b);
    }
}
```

```
$c = divide(5,0);
var_dump($c);
```

这样一来，当参数$b 为 0 时，系统返回 FALSE。那么变量$c 就成了一个布尔型变量，它的值为 FALSE。

8.1.3　局部变量、全局变量和静态变量

我们知道了所谓的参数其实就是变量。不过，这种变量与普通变量不同，它常常被叫做局部变量，只在函数内部起作用。与之相对的，就是全局变量了。所谓全局变量，顾名思义，其实就是在函数内部定义的，在脚本中的任何地方都有效的变量。

至于静态变量，就有些不太好理解了。变量之所以叫变量就因为它是活动的，可以变化的。静态的变量，听上去就有些不可思议，一个可以变化的东西怎么能叫静态呢？其实，静态变量翻译成英语为 static variable。在英语里，如果我们说一个东西是 static，就是说它在任何环境里都不会动或者不会改变。而 variable 本身却是可以改变的。那么用 static 来修饰 variable 就是说，静态变量的值在相应的函数中一旦被定义，在下次使用该函数时静态变量的值应该为上次修改后的值。它的值在同一个作用域中都会保持不变。

例 8.4 定义了三个函数。在这三个函数中，我们分别使用了局部变量、全局变量和静态变量。来看看这些函数返回的值有什么差别吧。

【例 8.4】 局部变量、全局变量和静态变量的使用。

```
<?php
    //函数 testLocal()中的变量$temp1 为局部变量
    function testLocal(){
        $temp1 = 5;
        $temp1 = $temp1 * 2;
        return ($temp1);
    }

    testLocal();
    echo "\$temp1 is ".$temp1;
    echo "<br>";
    testLocal();
    $secondRun = testLocal();
    echo "\$temp1 is ".$secondRun;
    echo "<br>";

    // 函数 testGlobal()中的变量$temp2 为全局变量
    function testGlobal(){
        global $temp2;
        $temp2 = 5;
        $temp2 = $temp2 * 2;
        return ($temp2);
    }

    testGlobal();
    echo "\$temp2 is ".$temp2;
    echo "<br>";
    testGlobal();
    $secondRun = testGlobal();
    echo "\$temp2 is ".$secondRun;
```

```
    echo "<br>";

    //函数 testStatic()中的变量$temp3 为静态变量
    function testStatic(){
        static $temp3 = 5;
        $temp3 = $temp3 * 2;
        return ($temp3);
    }

    testStatic();
    echo "\$temp3 is ".$temp3;
    echo "<br>";
    $secondRun = testStatic();
    echo "\$temp3 is ".$secondRun;
    echo "<br>";
?>
```

在这段长达 46 行的脚本中，定义了三个函数，它们分别是 testLocal()、testGlobal()和 testStatic()。在这三个函数里，分别定义了三个变量：$temp1、$temp2 和$temp3。随后分别将这三个变量与 2 的乘积赋给了它们自己。然后，每个函数运行两次：第一次运行后，输出当前函数对应的变量的值；第二次运行后，输出当前函数的返回值。

这段脚本运行的结果如图 8-4 所示。

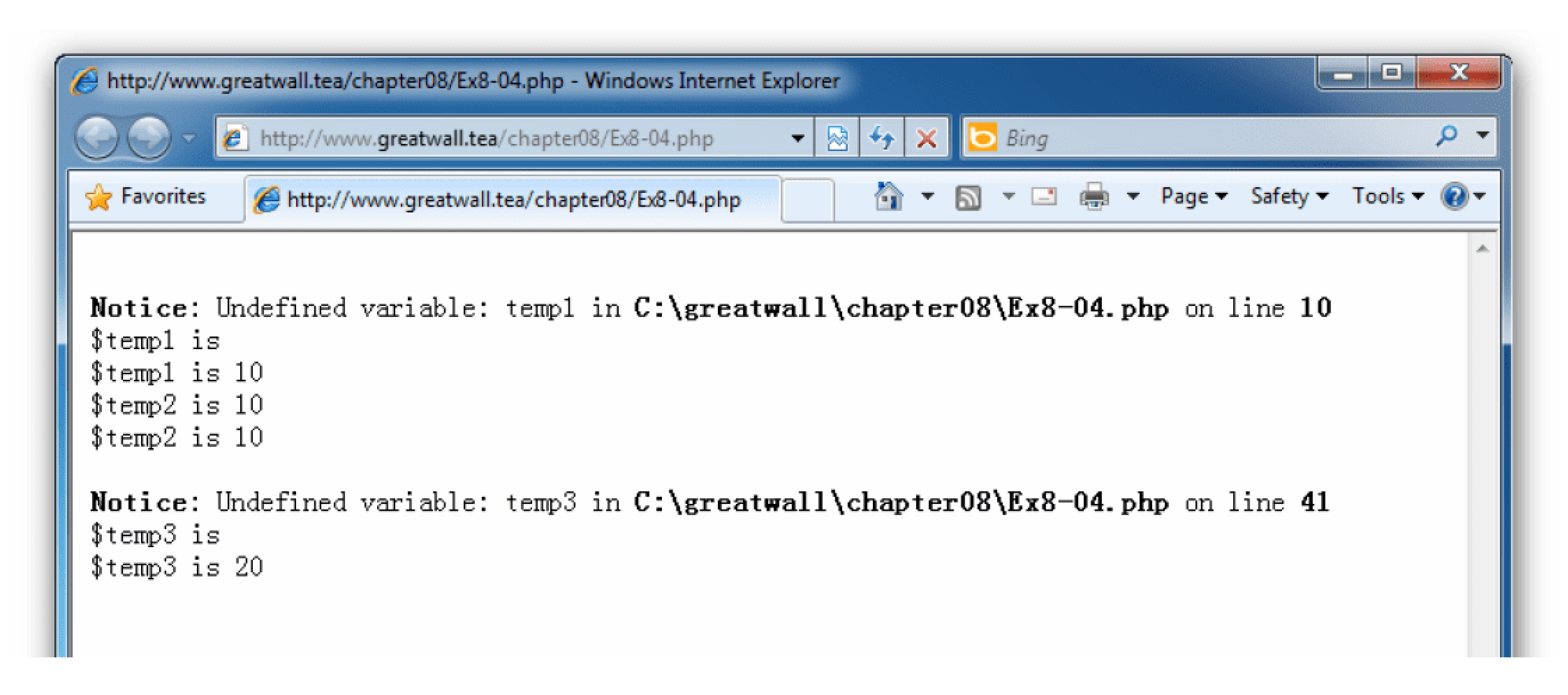


图 8-4　局部变量、全局变量和静态变量

我们知道，函数 testLocal()中定义的变量$temp1 为局部变量，它只在这个函数内部有效，当在函数外部使用这个变量时，系统会提示这个变量不存在。我们只能通过函数 testLocal()返回的值来得到变量$temp1 的值。因此，第一次运行 testLocal()函数后，输出的内容为“temp1 is ”，而第二次运行 testLocal()函数后，输出的内容为“temp1 is 10”。

在第二个段落里，函数 testGlobal()中定义的变量$temp2 为全局变量，因为在定义变量$temp2 时使用了 global 关键字。值得注意的是，在定义全局变量时不能同时给变量赋值。因此在 testGlobal()函数中另起了一行对变量$temp2 赋值。全局变量需要在函数内定义，但在脚本中的任何地方都有效。因此，第一次运行 testLocal()函数后，输出的内容为“$temp2 is 10”，而第二次运行 testLocal()函数后，通过输出该函数的返回值得到的结果依然为“$temp2 is 10”。

在第三个段落里，函数 testStatic()中定义的变量$temp3 为静态变量，因为在定义变量

$temp3 时使用了 static 关键字。与 global 关键字不同的是，在使用 static 定义静态变量的时候，我们是可以同时给变量赋值的。需要注意的是，静态变量跟局部变量一样，都只在函数内部有效。因此，在第一次运行 testStatic()变量后，系统提示变量$temp3 未定义。而第二次运行 testStatic()变量后，系统输出的结果表明变量$temp3 的值经过三次函数的使用变成了 20。

例 8.4 这段脚本和上面这三段话消化起来不太容易，我们小小地总结一下：

- 局部变量只能在函数内定义，只在定义它的函数内有效。
- 全局变量只能在函数内定义，在定义它的函数内外都有效。
- 静态变量只能在函数内定义，只在定义它的函数内有效，其值一旦改变，在下次改变前保持上次修改后的值不变。

这三种变量中，只有静态变量不太好理解。这样，再出个小题考考大家，看看大家对静态变量掌握的如何。

假若去掉例 8.4 第 43 行中的“$secondRun =”，只保留 testStatic()函数本身。然后把第 44 行中的“$secondRun”换成“$testStatic()”。那么第 44 行输出的结果是什么呢？为什么呢？

脑袋转的快的同学估计已经想到了：输出的结果应该为“$temp3 is 40”，因为 testStatic()函数又被运行了一次。上次运行后变量$temp3 的值已经变成 20 了。当再一次运行该函数后，变量$temp3 的值自然就变成 40 了。

想通了吗？☺

8.1.4　引用外部变量

在定义函数时，除了可以使用上面提到的局部变量、全局变量和静态变量之外，还可以引用外部变量。所谓外部变量是相对于局部变量来说的。与全局变量不同的是，外部变量是在函数外定义的，也只在函数外使用。

在这一小节里，我们先看一段脚本，然后来谈谈如何在函数中引用在函数外部定义的变量。

【例 8.5】 在函数中引用外部变量。

```
<?php
    /* double1()函数内的变量$varInt 是一个局部变量，
    而在该函数外部定义的同名变量是一个外部变量 */
    function double1($varInt){
        $varInt = $varInt * 2;
    }

    $varInt = 5;
    double1($varInt);
    echo "\$varInt1 is ".$varInt;

    //在 double2()函数内的变量$varInt 是对在该函数外部定义的同名变量的引用
    function double(&$varInt){
        $varInt = $varInt * 2;
    }

    $varInt = 5;
```

```
    double2($varInt);
    echo "\$varInt2 is ".$varInt;
?>
```

这段脚本运行的结果如图 8-5 所示。

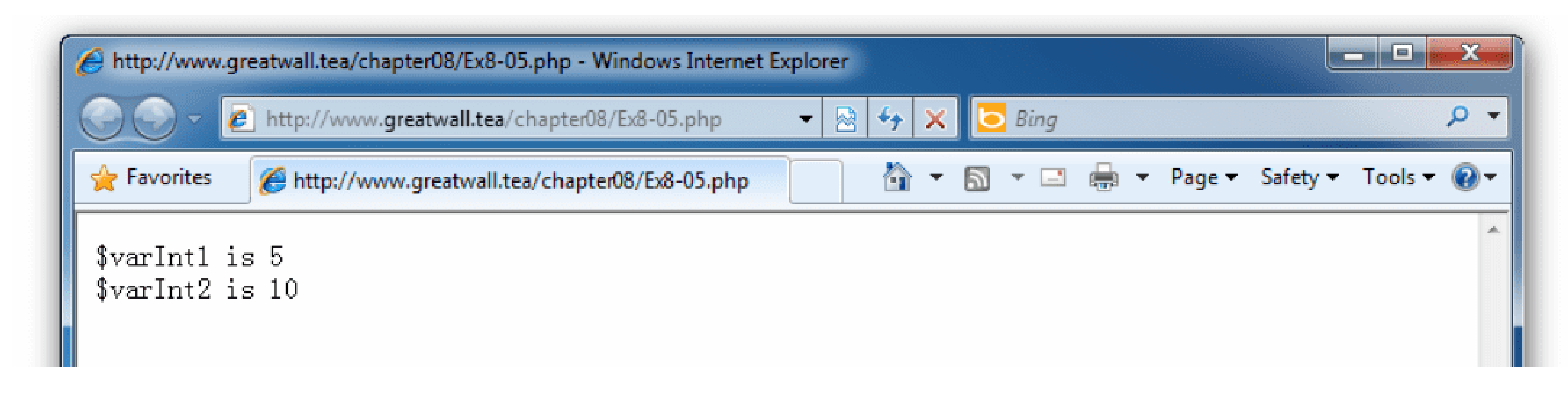


图 8-5　在函数中引用外部变量

在这段脚本中，定义了两个函数，它们分别是 double1()和 double2()。

在 double1()函数中，定义了一个局部变量$varInt，并在函数中将它与 2 的乘积赋值给它自己。然后在 double1()函数外又定义了一个同名变量，并为其赋值 5。接着执行 double1()函数。最后输出变量$varInt 的值。根据图 8-5 显示的结果可知，在函数外定义的变量$varInt 的值未发生变化。

在 double2()函数中，我们在定义函数时使用了一个引用变量。注意在$varInt 前面加了一个“&”符号，表示引用。除此之外，double2()函数内外的脚本与 double1()函数是一模一样的。可是，根据图 8-5 显示的结果，在函数外定义的变量$varInt 的值变成了 10，而这正是根据函数内的那个表达式可以得出的结果。可是为什么呢？

所谓引用，其实就是把这个外部变量的值代入函数内进行运算。当运行 double2()函数时，我们把外部变量$varInt 的值引入了函数中，这时函数内的表达式中使用的引用变量就相当于这个外部变量本身。所以$varInt2 的值就变成了 10。

这就是变量的引用。

8.1.5　函数的引用

如果函数只能在定义它的脚本文件中使用那就太逊色了。PHP 提供了 include 和 require 两种方法用于在一个脚本中引用其他脚本中定义的函数。确切地说，是引用定义这些函数的脚本文件。

还记得在“8.1.1　小试牛刀”中定义的 bold()函数吗？如果我们想在其他的脚本中也使用 bold()函数，就可以把 bold()函数的定义脚本写入一个叫自定义文件名的脚本文件（比如 function.inc）中。然后在需要使用 bold()函数的脚本文件开始，写上下面这句话：

```
include "function.inc";
```

注意，这时文件 function.inc 必须与写入这句话的脚本文件在同一个目录中，文件 function.inc 才能够被成功引用。在实际编程中，我们通常会把被引用的文件都放在一个叫 inc 的文件夹中，然后使用相对路径来引用它们。假设把刚才创建的 function.inc 文件放入了 inc 文件夹中，这个 inc 文件夹与需要引用 function.inc 的脚本文件层级相同，那么可以使用下面这句话：

```
include "../inc/function";
```

但是如果日后引用了 function.inc 的脚本文件换了地方，就得修改脚本了。为了避免出现这种尴尬的情况，也可以使用绝对路径来引用 function.inc。假设需要引用 function.inc 的脚本文件在网站的根目录下，而包含 function.inc 的 inc 文件夹在网站的根目录下的 lib 文件夹中。我们可以使用下面这句话：

```
include "/lib/inc/function";
```

这样一来，无论我们引用了 function.inc 的脚本文件放在什么地方，都不用再修改脚本了。

一般来说，我们使用 require 的时候比使用 include 的时候多。因为一旦被引用的脚本有错误，使用 include 会导致系统警告，而使用 require 则可以让系统直接终止被引用脚本，不发出任何警告。

还有，我们可以在一个脚本中引用多个文件，也可以在被其他脚本文件引用的文件中再引用其他的文件。这样虽然很方便，但可能造成某些文件被 include 或 require 了多次。为了解决这个问题，PHP 还提供了 include_once 或 require_once 这样两条语句。使用它们，就不怕重复引用了。

8.2　类

在本章的开始，我们用交通警察打了个比方，说明了函数、类和对象之间的关系。上一节，系统的了解了什么是函数。本节，我们就好好地讲解讲解什么是类、什么是对象。

在 PHP 中，所谓“类”，指的是一个包含一些数据和如何访问与修改这些数据的方法的包。这些数据就是一些变量，不过在类里，它们被称做“属性”。而访问与修改这些数据的方法就是一些函数，只是在类里才会被叫做“方法”。

当定义好一个类之后，并不能直接读取和使用类中定义的“属性”和“方法”，而是要通过创建一个基于这个类的对象来使用它。这个创建对象的过程就叫做“实例化”。读者可以把类想像成一个模具，而把对象想象成用这个模具做出来的复制品。一个模具可以用来做成千上万个复制品，而一个类也可以被实例化成无数个对象。

如果写了一个类，你自然是知道类的各种方法是如何实现的。但是这并不意味着你必须知道一个类中定义的各种方法是如何实现的，才可以使用这个类。因此，你可以在你的脚本中使用已经定义好的类来创建对象，进而使用类中定义的属性和方法。

8.2.1　如何定义类

例 8.6 定义了一个叫 Arithmetic()的类。我们一起来看看这个类是怎么写的。

【例 8.6】 简单算术类的使用。

```
<?php
    class Arithmetic {
        //定义两个成员变量
        var $a = 0;
        var $b = 0;

```

```
        /定义一个让两成员变量相加的方法
        function addUp(){
            return $this -> a + $this -> b;
        }

        //定义一个让两成员变量相减的方法
        function minusDown(){
            return $this -> a - $this -> b;
        }

        //定义一个让两成员变量相乘的方法
        function timesUp(){
            return $this -> a * $this -> b;
        }

        //定义一个让两成员变量相除的方法
        function divideDown(){
            return $this -> a / $this -> b;
        }
    }

    //定义一个基于 Arithemtic 类的对象$doMath
    $doMath = new Arithmetic;

    //为对象$doMath 的两个属性赋值
    $doMath -> a = 6;
    $doMath -> b = 3;

    //打印一些结果
    echo "6 + 3 = ".$doMath -> addUp();
    echo "<br>";
    echo "6 - 3 = ".$doMath -> minusDown();
    echo "<br>";
    echo "6 x 3 = ".$doMath -> timesUp();
    echo "<br>";
    echo "6 / 3 = ".$doMath -> divideDown();
?>
```

这段脚本创建了 Arithmetic 类，并在这个类中定义了两个成员变量和四个成员函数。用专业的角度来说，应该是在一个类中定义了两个属性和四个方法。定义类时需要使用关键字“class”，就像下面这样：

```
class className {

}
```

在类中定义成员变量时，前面得加上关键字“var”，这时可以选择为其赋值。在这里赋值的目的除了让成员变量有个初始值之外，也可以指定成员变量的变量类型。比如在这里为成员变量$a 和$b 赋值 0，表明它们都是整型变量。

在类中定义成员函数时，若在函数中需要使用成员变量的，需要通过特殊变量“$this”来引用。这个特殊变量相当于一个占位符，用来替代还没有创建的对象。一旦创建了基于这个类的某个对象，我们就可以使用“$*objectName* ->”来代替“$this ->”访问这个对象的属性和方法了。

另外，在成员函数中用到的成员变量是不需要在函数名后面的括号中事先声明的。这与定义普通函数有些不同。假设，我们要定义一个普通函数用来让两个数相加，得写成下

面的样子：

```
function addUp($a, $b){
    return ($a + $b);
}
```

而在类中，直接让函数名后面的括号空着就好，除非你需要添加其他的参数。

在定义好 Arithmetic 类之后，就可以创建基于这个类的对象了。在例 8.6 中，我们使用关键字“new”创建了一个基于 Arithmetic 类的对象$doMath。然后使用“$doMath ->”定义了该对象两个属性的值。注意，通过“->”访问对象的属性时，属性前的“$”一定要去掉。最后，使用“$doMath ->”分别引用了类的四种方法来输出两个属性的值相加、相减、相乘和相除的结果。

类和普通函数一样，可以与其他脚本分离开来。我们可以把所有的类都存放在一个独立的文件里，然后通过 include、require、include_once 或 require_once 来引用这个文件，从而使用文件中存放的类。

8.2.2　魔术方法__construct()和__destruct()

在 PHP 5 中，有一些变量叫做魔术变量，有一些方法被叫做魔术方法。这么叫不是因为它们会变魔术，而是因为它们可以很方便地帮助我们获取一些在之前的 PHP 版本里很难获取的数据和使用一些可以让 PHP 类变得更加易用的方法。值得注意的是，所有的魔术变量和魔术方法前面都会有两道下划线（__）。

在这一小节里，我们将讨论一下__construct()和__destruct()方法，这两种方法在英语里被称作 constructor 和 destructor，其词根都是“or”，表明它们是用来做某件事情的人或工具。这里显然把它们当成工具更合适。注意，这两种方法（注意我们用的词是“方法”）只能用在 PHP 类中，不能用在其他地方。它们的功能与 PHP 类也有着密切的关系：

- ❑ 在类被“实例化”的过程中，__construct()方法中的代码会被自动执行。
- ❑ 在类被“销毁”时，__destruct()方法中的代码也会被自动执行。

在例 8.7 中，定义了一个“汽车”类（Car），然后为这个类定义了一个属性“汽油”（$gas）和一个方法“加油”（addGas()），然后将这段脚本另存为文件 class.inc.php。

本节例 8.7 至例 8.14 是一个完整的实例，其中代码行号不连续是因为空行行号没给出。

【例 8.7】 魔术方法__construct()的使用。

```
<?php
    //定义一个类并为其命名 Car
    class Car {
     //define a property named gas
        var $gas = 0;
        var $addAmount = 0;

        //定义一个方法并为其命名 addGas
        function addGas($amount = 0){
            if (is int($amount)){
                if ($amount + $this -> gas <= 50) {
                    $this -> gas = $this -> gas + $amount;
                    $this -> addAmount = $amount;
                    return ("$amount gallons are added to your gas tank.");
```

```
15                      } else {
16                          return ($this -> gas." gallons are already in
                            your gas tank.<br>
17                          You can add ".abs(50 - $this -> gas)." gallons
                            more.");
18                  }
19              } else {
20                  return "Excuse me! An integer is required.";
21              }
22          }
23
24          //定义__construct()方法，初始化油箱现存油量
25          function __construct(){
26              $this -> gas = 50;
27          }
28      }
29  ?>
```

在这个 Car 类中，使用了魔术方法__construct()，而且在里面为“$this”的汽油属性定义了一个初始值 50。

然后，新建一个脚本文件 Ex8-07.php，并在文件中写入如下所示的脚本：

```
1   <?php
2       require_once "class.inc.php";
3
4       //创建一个基于 Car 类的对象$fordFiesta
5       $fordFiesta = new Car;
6
7       //显示 Ford Fiesta 现存油量
8       echo "Gas Level: ".$fordFiesta -> gas." gallons<br><br>";
9
10      //给 Ford Fiesta 加油
11      echo $fordFiesta -> addGas(10);
12  ?>
```

这段脚本运行的结果如图 8-6 所示。

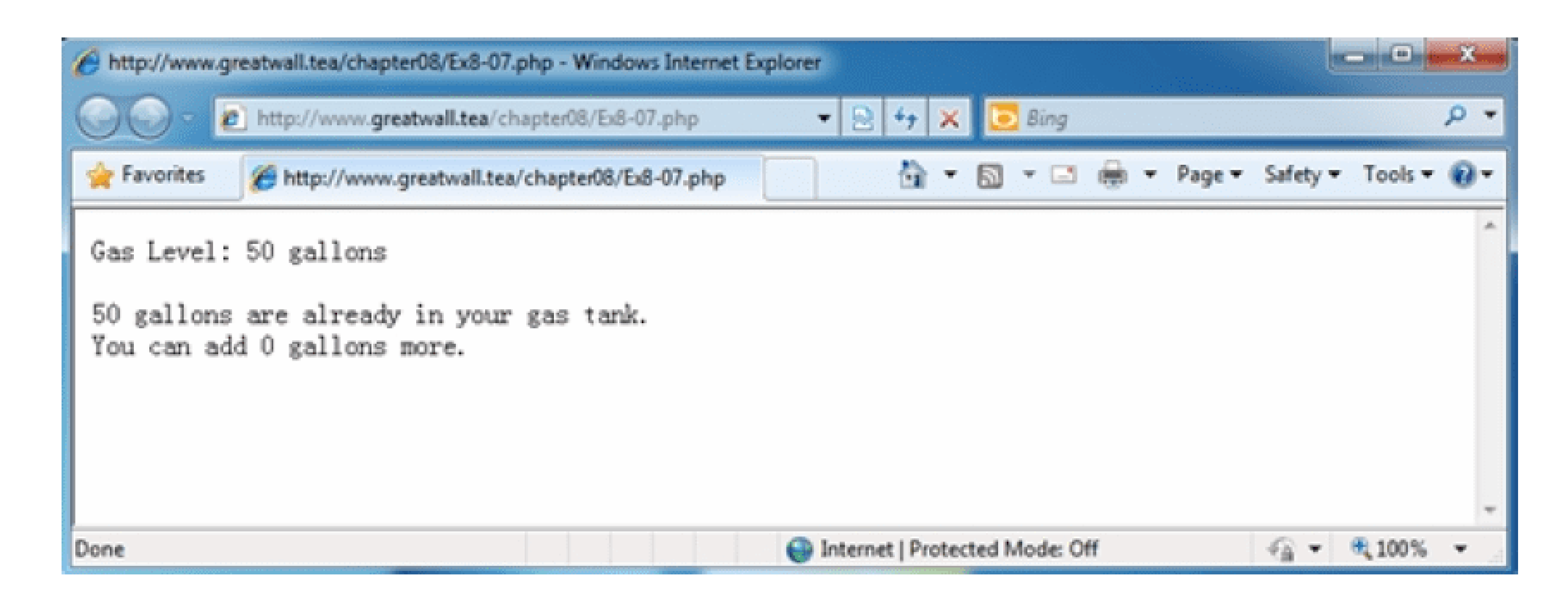


图 8-6　魔术方法__construct()

要知道，在 class.inc.php 的第 5 行，为属性$gas 赋的值是 0，而在 Ex8-07.php 的第 8 行，是直接输出“$fordFiesta -> gas”的值。而在这之前，我们没有使用 Car 类的任何方法来改变这个属性。唯一的解释只能是__construct()方法中的代码在对象$fordFiesta 被创建的同时被运行了。在 __construct()方法里，我们使用了“$this -> gas = 50”这样的语句，而“$this”指代的就是诸如$fordFiesta 这样的基于 Car 类的对象。因此，属性$gas 的值才会由 0 变成 50。

之后，在 Ex8-07.php 的第 11 行，执行了对象$fordFiesta 的 addGas()方法，并为其指定了一个参数：需要添加的汽油量。注意，在定义这个 addGas()方法时，我们指定了参数$amount，并为其指定了一个默认值 0。这样一来，如果在使用 addGas()方法时，没有指定参数值，系统就会自动使用这个默认值，从而保证脚本不会报错。另外，在 addGas()方法内，我们使用了两层 if…else 语句的嵌套：首先判断了$amount 是否为一个整数，若不是，则提示输入一个整数；然后判断了$amount + $this -> gas 的值是否超过 10，若不是，则提示油箱已满，无需加油。

大家可以尝试一下，把脚本第 11 行修改成如下的样子：

```
11      echo $fordFiesta -> addGas();
```

或

```
11      echo $fordFiesta -> addGas("Hello!");
```

看看会输出什么结果？和你想的是否一样呢？

与__construct()方法中的语句会在一个类被实例化时得到执行一样，如果我们在类中定义了__destuct()方法，这个方法中的语句也会在类被销毁时被执行。但与__construct()类不同的是，__destruct()方法不允许携带任何参数。

那么什么时候才需要使用__destruct()方法呢？通常情况下，我们一般不需要使用这个方法。不过如果你想在某个类被销毁时做一些清扫工作，也可以加上它。另外，如果一段脚本需要调试，也可以把__destruct()当成一个调试工具。

在后面的章节中，会看到关于__destruct()方法的示例。因此，在这里就不再赘述了。

8.2.3　类的继承

在前面的小节里，我们讨论了类与对象的关系。对象与类之间只存在“复制”关系，对象是类的具体化和实例化，而类是众多对象的抽象化。和类与对象的关系不同的是，继承是类与类之间的一种关系。比如，我们有一个类叫 Car，在这个类中，定义了一些所有汽车都会有的通用属性和方法。然后又定义了一个类叫 FordCar，而这个类中除了拥有所有在 Car 类中定义的属性和方法之外，还新增了一些只有 FordCar 类才有的属性和方法。那么 FordCar 类和 Car 类之间的关系就存在着继承与被继承的关系。被继承的一方叫做“父类”，而继承的一方叫做“子类”。

父类和子类间的关系就有点像父子关系，儿子会继承父亲的某些特点，但又与父亲是完全不同的两个人。类也一样，父类有的属性，子类可能都会有，而子类有的属性，父类却不一定有。

我们应该如何使用 PHP 的语言来表述这种关系呢？先来看一段脚本。

【例 8.8】 类的继承。

还是先在 class.inc.php 中写入如下脚本：

```
//定义一个继承自 Car 的子类并为其命名 FordCar
class FordCar extends Car {
    //Define some attributes
    var $gasConsumption = 1.6;
    var $avgSpeed = 50;
```

```

        function timeCalc(){
            $timeToDie = ($this -> gas / $this -> gasConsumption) / $this
            -> avgSpeed;
            return ("You can drive no more than $timeToDie hours.");
        }
    }
```

然后新建一个脚本文件 Ex8-08.php，并写入如下脚本：

```
<?php
    require_once "class.inc.php";

    $fordFox = new FordCar;

    echo "Gas Level: ".$fordFox -> gas." gallons.<br><br>";
    echo $fordFox -> timeCalc();
    echo "<br>";

    echo $fordFox -> addGas(10);
?>
```

在例 8.8 的前一段脚本中，定义了一个类 FordCar。值得注意的是，在定义 FordCar 这个类时，使用了 extends 关键字，表示 FordCar 这个类是从 Car 这个类继承来的。在 FordCar 这个类中，定义了一个方法 timeCalc()来估算按照现存油量，车还能走多久。在 timeCalc()方法中，使用了在 Car 类中定义的 gas 属性。

然后在创建 FordCar 类之后，在 Ex8-08.php 这个文件里创建了一个基于 FordCar 类的对象，并为其命名$fordFox。接着输出了这辆车的现有油量、预计可行驶时间和执行加油的结果。在输出这些信息时，我们使用了从 Car 类继承过来的 gas 属性和 addGas()方法。

这段脚本输出的结果如图 8-7 所示。从图中可以看出，脚本运行正常，没有报错。

在这个例子中，Car 是 FordCar 的父类，而 FordCar 是 Car 的子类。FordCar 继承了 Car 中定义的所有属性和方法，并且在 Car 原有属性和方法的基础上添加了新的属性和方法。

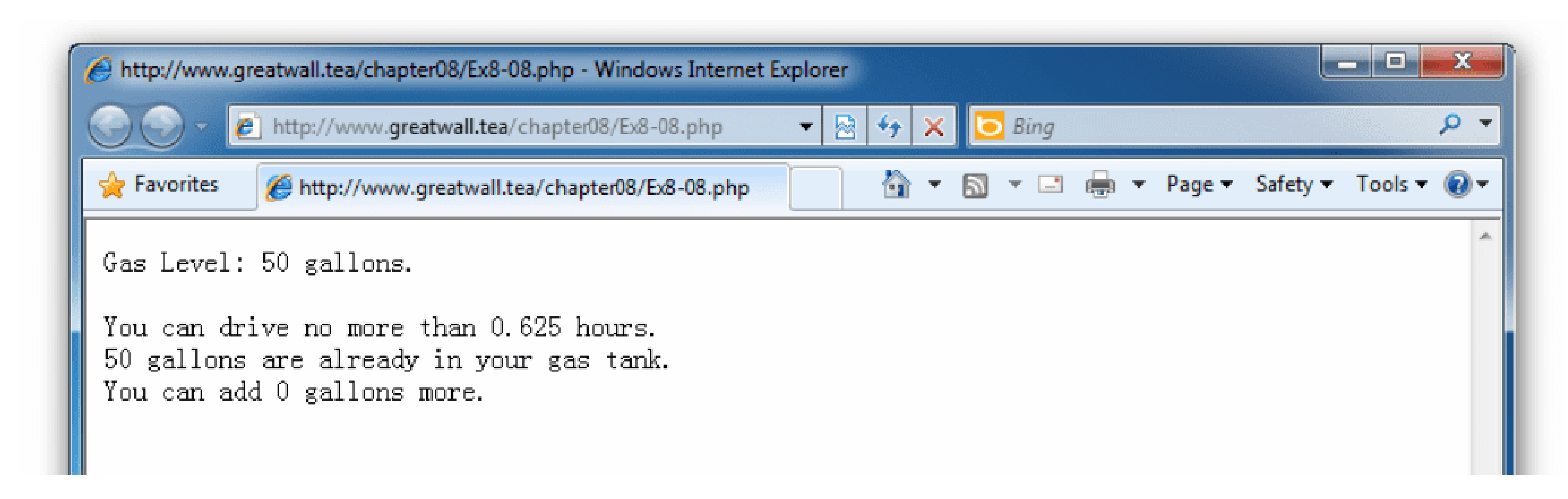


图 8-7　类的继承

除了全盘继承，还可以在子类中修改从父类继承过来的属性和方法，也包括对父类中的__construct()方法的修改。我们来看下面这个例子。

【例 8.9】 继承并修改父类的属性和方法。

先在 class.inc.php 中定义一个新的类，并为其命名 GMCar。

```
//define another class extended from class Car
class GMCar extends Car {
    var $gasType = array ("Gas 89" => 7.31,
```

```
                        "Gas 92" => 7.81,
                        "Gas 95" => 8.32,
                        "Disl 0" => 7.78);

    function gasQuot(){
        if ($this -> addAmount > 0) {
            $returnString = "You need to buy ".$this -> addAmount."
            gallons of gas.
                        <br>You need to pay:<ul>";
            foreach ($this -> gasType as $type => $unitPrice) {
                $returnString = $returnString."<li>$".$unitPrice *
                $this -> addAmount.
                            " for ".$this ->addAmount." gallons of
                            ".$type.".</li>";
            }
            $returnString = $returnString."</ul>";
            return($returnString);
        } else {
            return("No gas fee");
        }
    }

    function   construct($gas){
        $this -> gas = $gas;
    }
}
```

接着新建一个脚本文件，并命名为 Ex8-09.php。然后写入如下脚本：

```
<?php
    require "class.inc.php";

    $GMPolo = new GMCar(40);

    $GMPolo -> addGas(10);
    echo $GMPolo -> gasQuot();
?>
```

这段脚本的运行结果如图 8-8 所示。

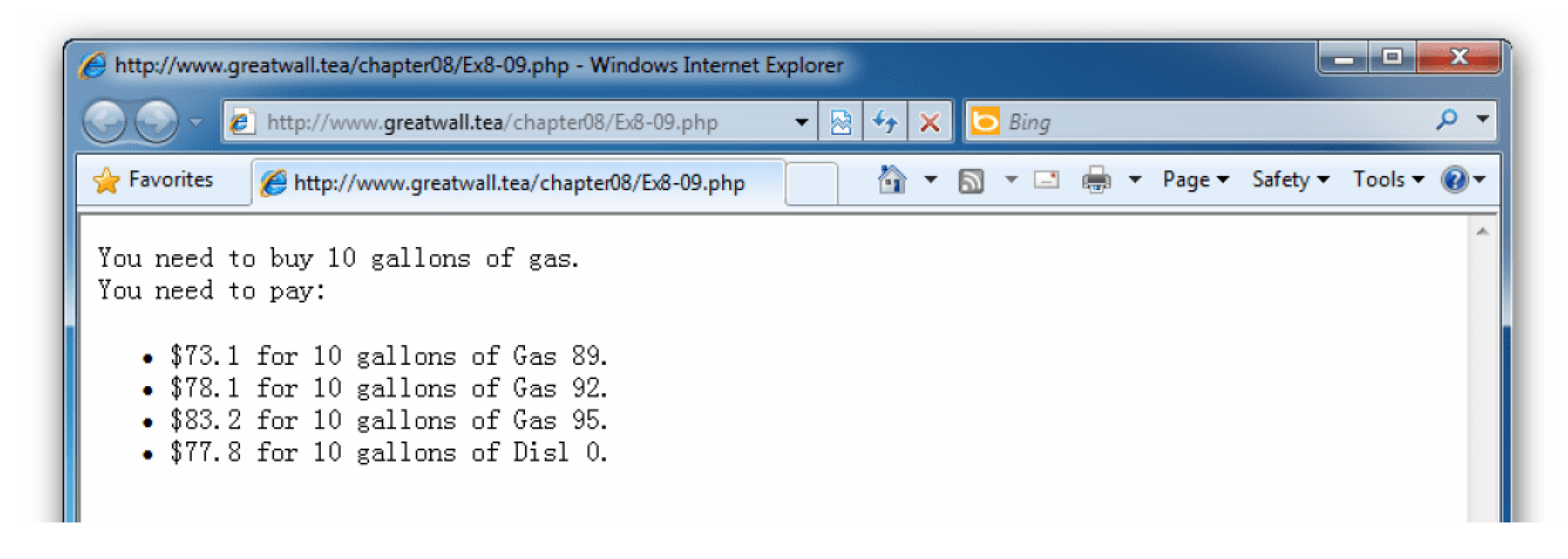


图 8-8　继承并修改父类的属性和方法

在例 8.9 所示中，我们重新定义了属性 gas 的初始值，而这个属性是 GMCar 类从 Car 类中继承的。在继承过程中，我们可以通过重新定义属性和方法来使这种继承关系发生某些变化。这与儿女与父母之间的继承关系类似，有部分基因会在继承过程中发生变异。

8.2.4　类的私有元素

与函数类似，我们可以在类中定义属于该类私有的属性和私有方法。通过定义私有属性和方法，就可以隐藏类的某些属性和方法，一定程度上保证了这些被隐藏的属性和方法不会在使用过程中被篡改。

为了说明这个问题，还是先回过头去看看例 8.7 中的 Car 类，所有的属性和变量都是可以在基于该类的对象中访问的。也就是说使用这个 Car 类，所有的属性都有可能被修改，所有的方法都有可能被使用。如果我们想隐藏某些属性和方法，可以定义这些属性和方法时使用 private 关键字，就像例 8.10 所示中一样。

【例 8.10】 类的私有属性。

```
class UnitCounter {
    private $units = 0;
    private $weightPerUnit = 1.0;

    function numberOfUnit (){
        return $this -> units;
    }

    function addUnit ($n = 1){
        $this -> units = $this -> units + $n;
    }
}
```

我们在 class.inc.php 中输入如上所示的这段脚本。在这段脚本中，定义了一个名为 UnitCounter 的类，这个类是用来计数的。在这个类中，定义了两个私有属性：一个是“物品个数（units）”，另一个是“物品单重（weightPerUnit）”。这两个私有属性是不可以在对象中直接访问的。然后定义了两种方法：一是“物品总数（numberOfUnit）”，另一个是“增加物品（addUnit）”。只有使用这两种方法才可以访问上面的两个私有属性。

接着新建 Ex8-10.php 文件，并在文件中输入以下脚本：

```
<?php
    require_once "class.inc.php";

    $apple = new UnitCounter;

   //下面两行脚本会报错
    echo $apple -> unit;
    echo $apple -> weightOfUnit;

    //下面两行脚本会正常执行
    $apple -> addUnit(10);
    echo $apple -> numberOfUnit();
?>
```

在这段脚本中，定义了一个基于 UnitCounter 类的对象$apple，然后使用$apple 对象来访问 unit 和 weightOfUnit 属性。接着又使用了 addUnit 方法增加了 10 只苹果，然后输出现有苹果的数量。正如脚本中备注说的一样，这段脚本的运行结果如图 8-9 所示。

通过输出结果，我们发现系统认定这两个属性没有定义。这也说明，我们可以通过定义私有属性来隐藏某些不想让类的使用者修改的内容。

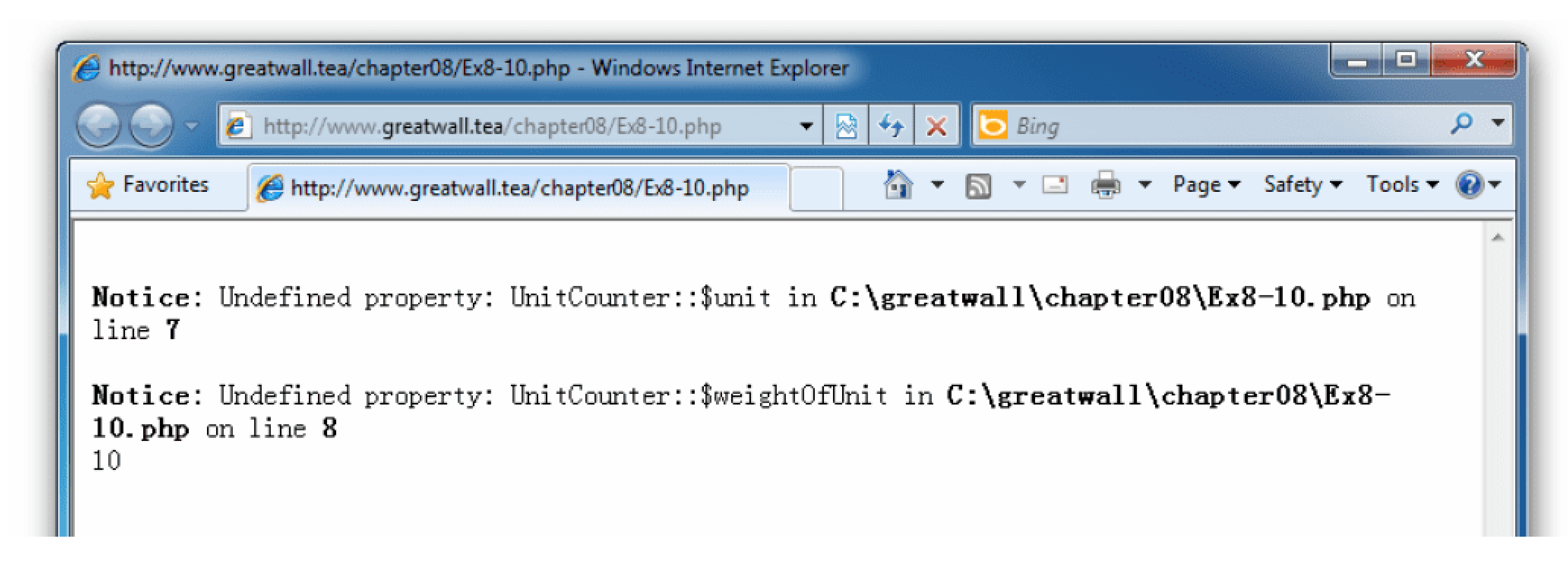


图 8-9　类的私有属性

私有方法与私有属性是在私密性方面是一样的。再来看一段脚本。例 8.11 这段脚本定义了一个 FreightCalculator 类，在这个类中，定义了两个私有变量：$numberOfCases 和 $totalWeight。然后又定义了两个私有方法：KgTotal()和 CaseTotal()。

【例 8.11】 类的私有方法。

我们还是在 class.inc.php 中定义 FreightCalculator 类。

```
class FreightCalculator {
    private $numberOfCases;
    private $totalWeight;

    function totalFreight (){
        return round($this -> CaseTotal() + $this -> KgTotal());
    }

    private function CaseTotal(){
        if($this -> numberOfCases > 5){
            return 5;
        } else {
            return $this -> numberOfCases * 1.0;
        }
    }

    private function KgTotal(){
        if ($this -> numberofCases > 5){
            return (($this -> totalWeight / $this -> numberOfCases) *
            ($this -> numberOfCases - 2))
                 * 0.2;
        } else {
            return 0;
        }
    }

    function   construct($numberOfCases, $totalWeight){
        $this -> numberOfCases = $numberOfCases;
        $this -> totalWeight = $totalWeight;
    }
}
```

在这个 FreightCalculator 类中，我们的计算公式是：若件数（numberOfCases）小于 5，则只按件计价，每件 10 元；若件数（numberOfCases）大于 5，则除去按件计价的部分后，按重计价。最后输出总运费。

接下来，在新建的 Ex8-11.php 中输入如下脚本：

```
<?php
    require once "class.inc.php";

    $expressFreight = new FreightCalculator(15, 200);

    //这句脚本可以运行
    echo "You need to pay $".$expressFreight -> totalFreight().".";

    //这句脚本会报错
    echo $expressFreight -> CaseTotal();
?>
```

Ex8-11.php 定义了一个对象$expressFreight，同时指定了“件数（numberOfCases）”为 15，“总重量（totalWeight）”为 200。最后输出总运费和总件数。

这段脚本的运行结果如图 8-10 所示。

通过截图可知，总运费正常输出，而总件数则输出异常。系统提示无法调用对象的私有方法。

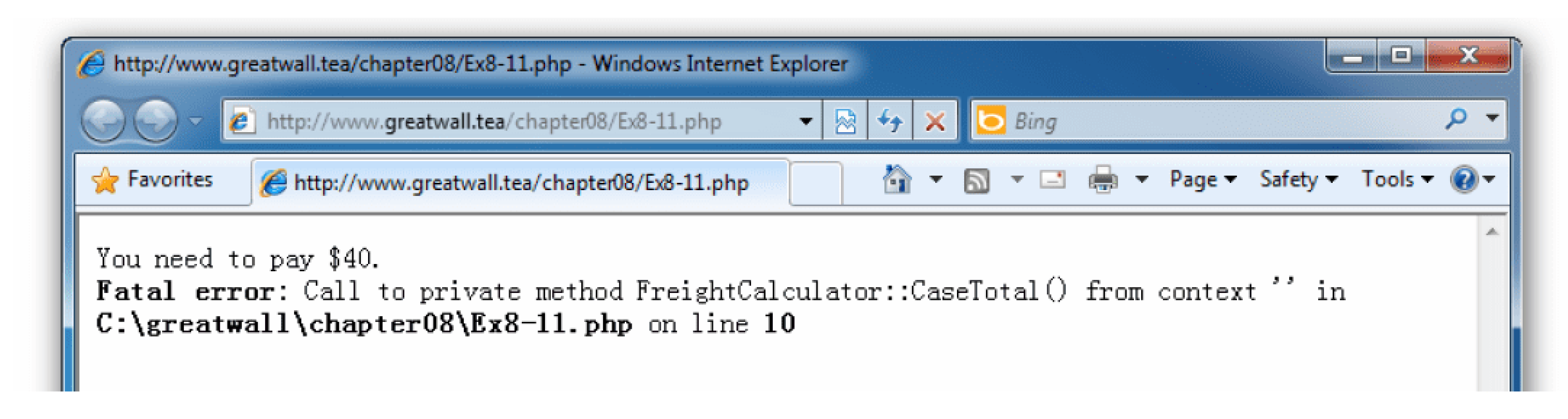


图 8-10　类的私有方法

8.2.5　类的静态元素

在类中，也可以把属性和方法定义为静态的。所谓的静态属性与普通属性不同之处在于，普通属性的值在每个基于某类的对象中是相互独立的，而静态属性的值在每个基于某类的对象中是共享的。也就是说，静态属性一旦在一个基于某类的对象中被改变了，那么在所有基于该类的对象中，这个静态属性的值也就都跟着改变了。

例 8.12 定义了一个 Revenue 类，用于统计一家商店某日的销售的产品类型及每类商品的销售额。这两个数据被存放在两个私有属性$commodityType 和$amount 中。Revenue 类同时还统计所有商品的销售总额和总计销售了多少种商品，这两个数据则被存放在了两个静态属性$totalRevenue 和$numberOfTypes 中了。值得注意的是，所有的基于 Revenue 类的对象都可以访问这两个静态属性并修改它们的值，但是与普通属性不同的是，在使用静态属性时，需要使用类名加双冒号（如 Revenue::）而不是之间使用$this 加箭头符号（如$this->）。此时，我们可以把这个类看成是个常量。

一起来看看这段脚本。

【例 8.12】 静态属性的使用

我们先在 class.inc.php 中定义 Revenue 类，脚本如下：

```
class Revenue {
    private $unitPrice;
```

```
    private $amount;
    private $unit;
    private $commodityType;

    static $totalRevenue = 0;
    static $numberOfTypes = 0;

    function info(){
        $revenuePerType = $this -> unitPrice * $this -> amount;
        $revenuePortion        =        round($revenuePerType        /
Revenue::$totalRevenue * 100);
        return $this -> amount." ".$this -> unit." ".$this ->
        commodityType.
               " have been sold. The revenue is $".$revenuePerType.", ".
               "which takes ".$revenuePortion."% of the total revenue" ;
    }

    function __construct($commodity, $price, $unit, $amount){
        $this -> commodityType = $commodity;
        $this -> unitPrice = $price;
        $this -> unit = $unit;
        $this -> amount = $amount;

        Revenue::$totalRevenue = Revenue::$totalRevenue + $this ->
        unitPrice * $this -> amount;
        Revenue::$numberOfTypes = Revenue::$numberOfTypes + 1;
    }

    function __destruct(){
        Revenue::$totalRevenue = Revenue::$totalRevenue - $this ->
        unitPrice * $this -> amount;
        Revenue::$numberOfTypes = Revenue::$numberOfTypes - 1;
    }
}
```

然后，新建 Ex8-12.php 文件并在这个文件中输入如下脚本：

```
<?php
    require once "class.inc.php";

    $dailyRecords = array(new Revenue("apples", 3.56, "kg", 320),
                      new Revenue("pears", 4.25, "kg", 400),
                      new Revenue("vaccume cleaners", 600, "", 25),
                      new Revenue("TV sets", 800, "", 100),
                      new Revenue("basketball", 60, "", 127));

    echo "<ul>";
    foreach ($dailyRecords as $dailyRecord){
        echo "<li>".$dailyRecord -> info()."</li>";
    }
    echo "</ul>";

    $total = Revenue::$totalRevenue;
    $count = Revenue::$numberOfTypes;
    echo "Total Revenue: $".$total."<br>";
    echo "Commodity Types: ".$count;
?>
```

在 Ex8-12.php 文件里的这段脚本中，定义了$dailyRecords 这个变量，然后把由五个基于 Revenue 类的对象组成数组赋值给它。接着使用了 foreach 语句输出了关于这几条交易记

录组成的报表。最后输出了总收入和产品类型总数。

这段脚本输出的结果如图 8-11 所示。

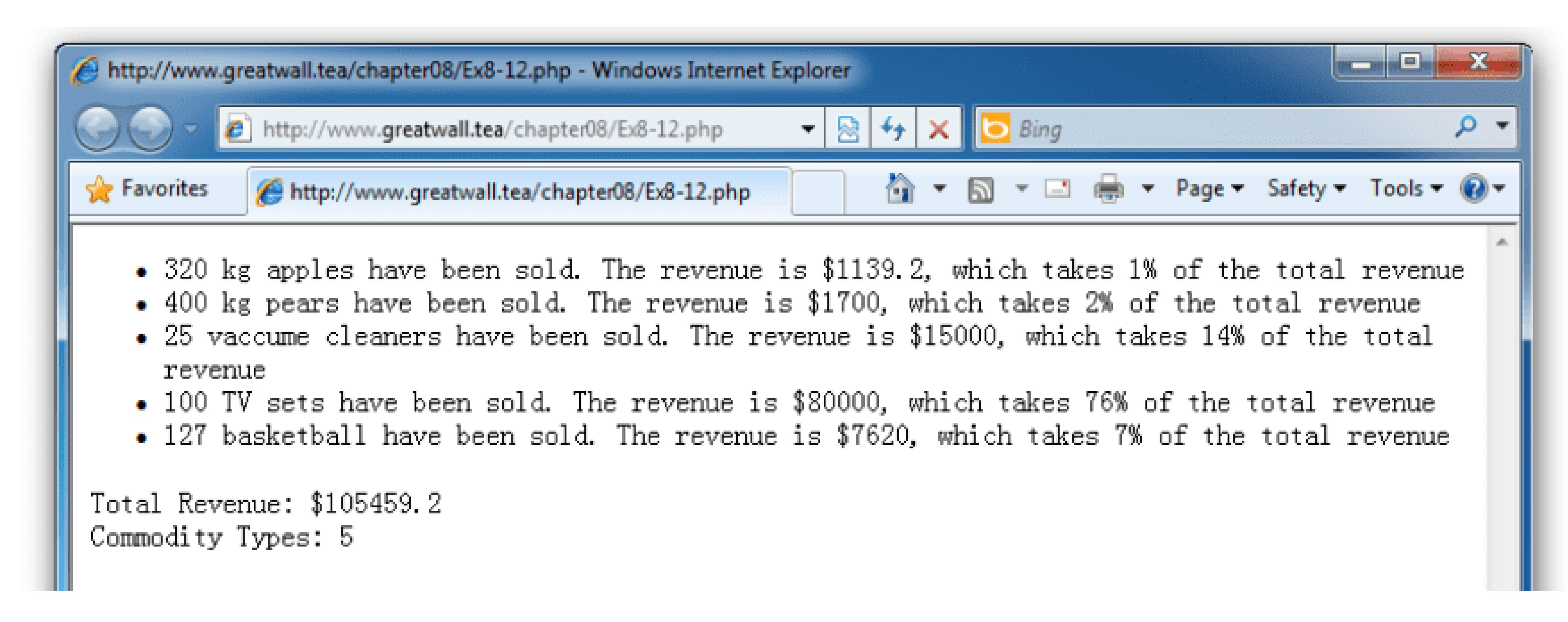


图 8-11　静态属性

如果仅需要对一个类的静态属性进行操作，那么可以不定义任何基于该类的对象直接修改静态属性的值。因为对访问静态属性使用的是“类名+::”而不是“$this->”。比如我们可以使用如下的脚本对 Revenue 类中的 totalRevenue 和 numberOfTypes 的值进行修改。

```
require once "class.inc.php";

//修改静态属性 totalRevenue 和 numberOfTypes 的值
Revenue::totalRevenue = 124000;
Revenue::numberofTypes = 20;
```

静态方法和静态属性是一个道理，都是在类这个层面的全局发挥作用。需要注意的是，静态方法只能访问静态属性，而不能访问基于该类的对象，因而也就不能使用$this 关键字。我们也可以在不定义任何基于某类的对象而直接访问该类中定义的静态方法。假如，在例 8.12 中的 Revenue 类里加上如下脚本：

```
static function total() {
    return Revenue::totalRevenue;
}

static function counts() {
    return Revenue::numberOfTypes;
}
```

然后我们可以使用如下的脚本访问这两个静态方法：

```
require once "class.inc.php";

//显示静态属性 totalRevenue 和 numberOfTypes 的值
echo Revenue::totalRevenue;
echo Revenue::numberofTypes;
```

8.3　对　　象

8.3.1　创建对象

在 8.2 节中，我们创建了若干个类，也基于这些类创建了若干个对象。在本小节里，

我们就来总结一下如何创建对象。

在例 8.7 中，我们定义了一个 Car 类，然后在 Ex8-07.php 中创建了一个基于 Car 类的对象$fordFiesta。在创建这个对象的时候，使用了 new 关键字。具体的做法是：

```
$fordFiesta = new Car;
```

在这一句脚本中，我们就像定义一个变量一样来定义对象。在定义这个对象时：

- 使用了变量定义符（$）来为对象指定对象名（$fordFiesta）；
- 使用了变量赋值符（=）来指定该对象所属类 （Car）。

值得注意的是，类名前是不使用变量定义符的。

在创建了对象之后，我们就可以通过这个对象使用该对象所属类的各种属性和方法了。至于如何使用这些属性和方法，你可以再回过头去看看前面学过的内容。

8.3.2　克隆对象

在定义一个类的时候，也可以使用魔术方法__clone()来定义当基于该类的某个对象被克隆时，该对象的属性和方法应该如何运作。如果在定义类时，没有使用__clone()这个魔术方法，则当基于该类的某个对象被克隆时，我们需要使 clone 这个关键字，而这个对象的所有属性和方法在被克隆的一刹那所处的状态都会原封不动的被克隆。

【例 8.13】 克隆对象的使用。

先在 class.inc.php 中写入如下脚本：

```
class GasFiller1 {
    private $gas = 100;
    private $operator = "SINOPEC";

    function sellGas($soldGas) {
        $this -> gas = $this -> gas - $soldGas;
        return $this -> gas." liters of gas are left. [".$this ->
        operator."]";
    }

    function __clone() {
        $this -> operator = "China Petrol";
    }
}

class GasFiller2 {
    private $gas = 100;
    private $operator = "SINOPEC";

    function sellGas($soldGas) {
        $this -> gas = $this -> gas - $soldGas;
        return $this -> gas." liters of gas are left. [".$this ->
operator."]";
    }
}
```

这段脚本定义了两个类 GasFiller1 和 GasFiller2 用来代表两台加油机。第一台加油机 GasFiller1 定义了两个私有属性$gas 和$operator 来代表加油机中现存油量和运营商。然后定义了 sellGas()方法用于售油。接着使用了魔术方法__clone()来实现当基于该类的对象被

克隆时修改加油机中的现在油量和运营商。

在第二台加油机 GasFiller2 中，除了没有定义__clone()方法之外，其他的脚本与 GasFiller1 一样。接着，我们新建 Ex8-12.php 文件，然后在该文件中写入如下脚本：

```
<?php
    require once "class.inc.php";

    $gasFiller1 = new GasFiller1;
    echo $gasFiller1 -> sellGas(20);
    echo "<br>";
    $gasFiller2 = clone $gasFiller1;
    echo $gasFiller2 -> sellGas(20);
    echo "<br>";
    $gasFiller3 = new GasFiller2;
    echo $gasFiller3 -> sellGas(20);
    echo "<br>";
    $gasFiller4 = clone $gasFiller3;
    echo $gasFiller4 -> sellGas(20);
?>
```

在这段脚本中，我们定义了四台加油机，它们分别为$gasFiller1、$gasFiller2、$gasFiller3 和$gasFiller4。其中，加油机$gasFiller2 和$gasFiller4 分别是$gasFiller1 和$gasFiller3 的克隆。我们先来看看这段脚本运行的结果如图 8-12 所示。

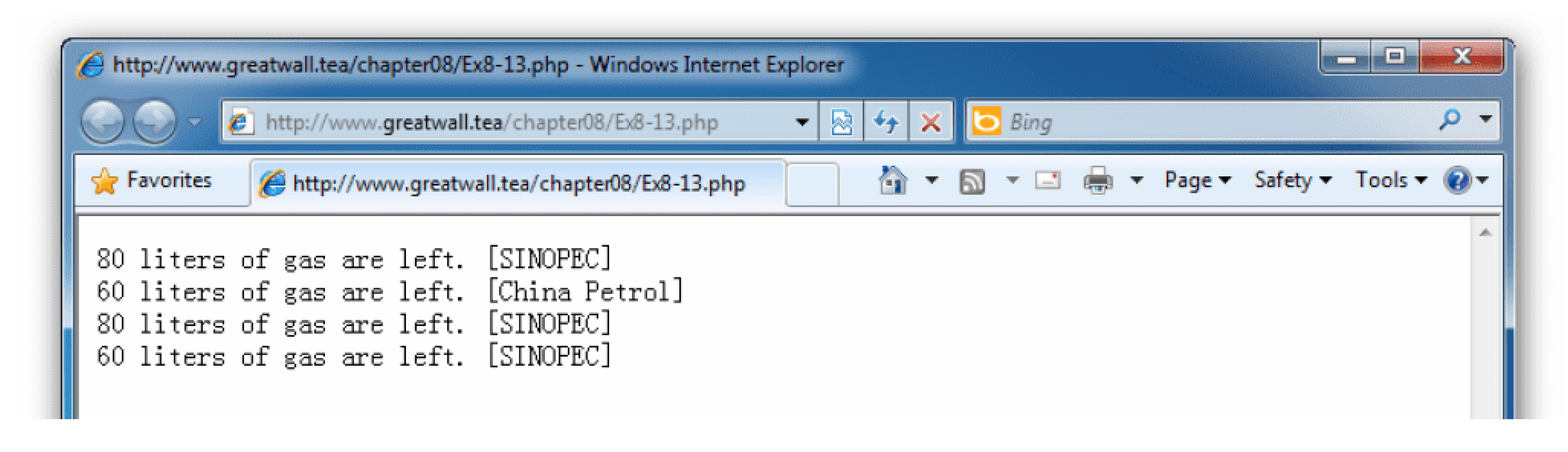


图 8-12　克隆对象

我们知道两个加油机类中的$gas 属性的初始值都是 100。通过图 8-12 所示的结果可以知道，加油机$gasFiller2 只克隆了$gasFiller1 中的现在油量，而重新定义了运营商，而加油机$gasFiller4 却克隆了$gasFiller3 的现存油量和运营商。

8.3.3　销毁对象

销毁一个对象，和销毁一个变量一样，都要使用 unset()方法。在 8.2.2 小节中，我们提到了魔术方法__destruct()。如果我们在定义某个类时定义了__destruct()方法，那么当使用 unset()方法销毁一个基于该类的对象时，__destruct()方法中定义的脚本会被执行。

【例 8.14】 销毁对象。

首先，在 class.inc.php 文件中添加一个 Calc 类。脚本如下：

```
    class Calc {
        private $a;
        private $b;

        function add(){
```

```
            return $this -> a + $this -> b;
        }

        function subtract(){
            return $this -> a - $this -> b;
        }

        function times(){
            return $this -> a * $this -> b;
        }

        function divide(){
            return $this -> a / $this -> b;
        }

        function __construct($a, $b){
            $this -> a = $a;
            $this -> b = $b;
        }

        function __destruct(){
            echo "the Calc is switched off...";
        }
    }
```

这个类一共有两个属性（$a 和$b）和四个方法（加、减、乘、除）。另外，还定义了两个魔术方法__construct()和__destruct()，前者用来赋值，后者用来提示基于该类的对象已销毁。

接下来，新建 Ex8-14.php，然后在文件中输入以下脚本：

```
<?php
    require_once "class.inc.php";

    $myCalc = new Calc(5, 10);
    echo $myCalc -> add()."<br>";
    echo $myCalc -> subtract()."<br>";
    echo $myCalc -> times()."<br>";
    unset($myCalc);
    echo $myCalc -> divide()."<br>";
?>
```

这段脚本定义了基于 Calc 类的对象$myCalc，并在定义$myCalc 对象时为属性$a 和$b 赋了值。然后我们分别输出了两个属性相加、相减和相乘的值，随后使用 unset()方法销毁了$myCalc 类。最后试图输出属性$a 除以属性$b 的值。这段脚本的运行结果如图 8-13 所示。

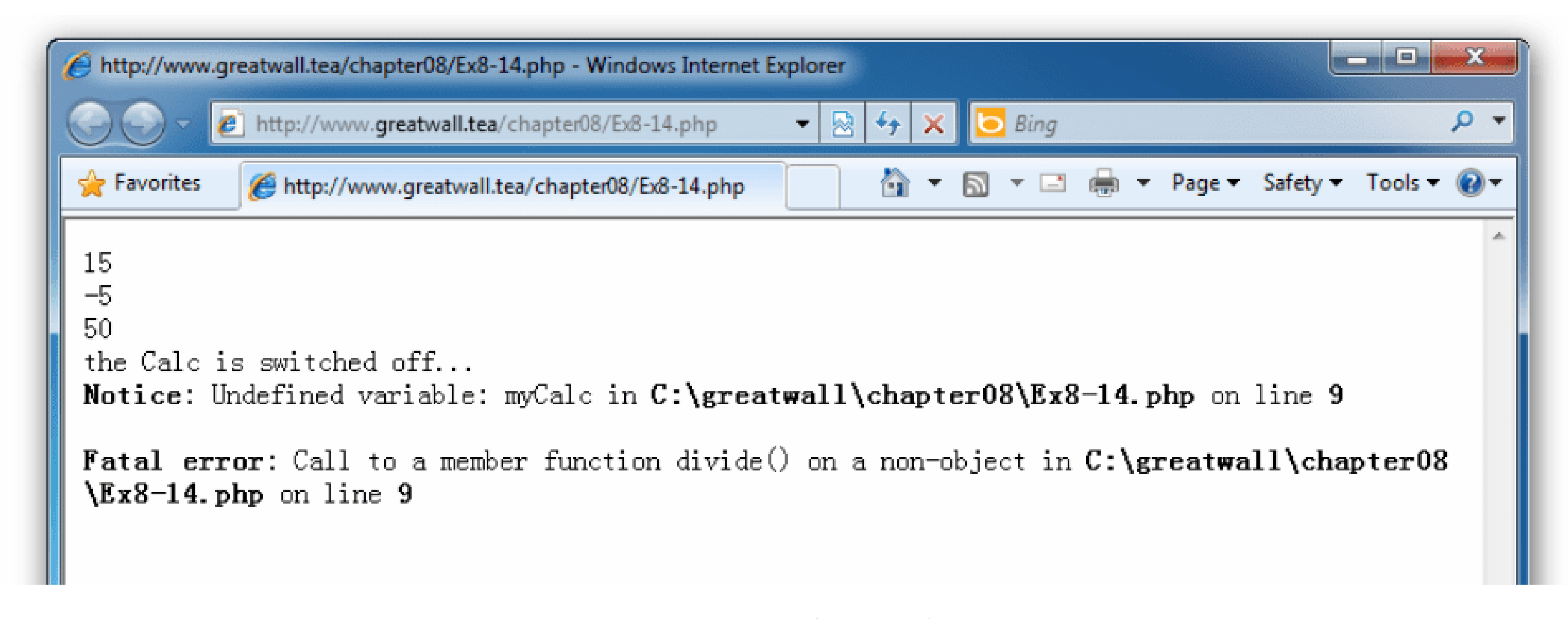


图 8-13　销毁对象

通过图 8-13 可知，当$myCalc 对象被销毁时，系统输出了如下的提示信息：

```
the Calc is switched off…
```

在$myCalc 对象被销毁之后，再试图使用该对象时，系统会提示我们$myCalc 既不是对象，也不是变量。

8.4　实战练习：记账工具（上）

在接下来的三个章节的实战练习中，我们将会动手制作一个记账工具。其中会用到在这三章中学习到的知识。

本节先来写一个记账工具类，为其定义一些属性和方法。

作为账务系统本身，它的基本功能应该有：单笔账务录入、批量账务录入、账务定期分类统计和账务格式化。同时，它也会有如下的属性：记账使用的货币单位和统计周期等。

对于每笔账务而言，它应该拥有以下属性：账务描述、账务产生时间、账务涉及的金额和账务所属分类。

基于上述分析，我们可以建立一个 bookKeeping 类，用来涵盖这些功能和属性。

```
<?php
    /**
     * 创建一个名为 bookKeeping 的类
     */
    class bookKeeping {


        // 初始化货币单位和统计周期
        function __construct() {
            $this -> unit = 'YUAN';
              $this -> cycle = 'MONTH';
        }

         // 检查并添加一条消费记录
         function AddSingleExpense($desc,$amount,$category,$userId){
              if(mb_strlen($desc,'UTF-8') <= 70 &&
                 intval($amount) > 0 &&
                 strlen($userId) > 0) {
                   $entryID = addExpenseToDatabase($desc,$amout,$category,
                   $userId);
                   return '添加消费记录 '.$entryID.' 成功.';
              } else {
                   return '参数不正确。添加消费记录失败';
              }
         }

         // 向数据库中添加一条消费记录
         private function addExpenseToDatabase($desc,$amount,$category,
         $userId){

         }


         // 检查并添加一条收入记录
```

```
        function AddSingleIncome($desc,$amount,$category,$userId){
            if(mb strlen($desc,'UTF-8') <= 70 &&
              intval($amount) > 0 &&
              strlen($userId) > 0) {
               $entryID = addIncomeToDatabase($desc,$amout,$category,
               $userId);
               return '添加收入记录 '.$entryID.' 成功.';
            } else {
               return '参数不正确。添加消费记录失败';
            }
        }

        // 向数据库中添加一条收入记录
        private function addIncomeToDatabase($desc,$amount,$category,
        $userId){

        }
    }

?>
```

上面的脚本定义了一个名为 bookKeeping 的类，它有两个属性，分别为“单位（unit）”和“记账周期”。除此之外，它还有若干方法分别用来添加消费和收入记录。值得注意的是，在类中，还定义了两个私有的方法，用于向数据库中添加记录。我们在实例化一个类之后，不能直接调用类的私有属性和方法，而只能通过类的公有方法来调用。在这个例子里，通过 AddSingleIncome()和 AddSingleExpense()方法分别调用了 AddIncomeToDatabase()和 AddExpenseToDatabase()这两个私有方法。

另外，因为这两个私有方法涉及到数据库的操作，我们将其留到第 10 章的实战练习中进行讲解。

8.5　习　　题

（1）请定义一个可以返回斜体文字的函数。

（2）请定义一个用于判断指定的数字是否为 100 以内的偶数的函数。

（3）请创建一个名为 styler 的类，然后在其中定义 bold()、italicize()方法。

（4）请创建一个继承自 styler 类的子类，并将其命名为 stylerEnhanced，然后在其中添加 underline()方法。

请创建一个水果分捡机（fruitPicker），用来分捡筐（basket）中的水果。筐中的水果如下：

```
apple,apple,apple,pear,pear,banana,apple,banana,pear,grapefruit,pear,ap
ple,banana,banana,banana
```

（5）创建一个名为 fruitPicker 的类，该类带有一个字符串类型的参数$basket：该类有两个私有属性（$fruitName 和$fruitQuantity）和一个静态属性（$totalQuantity）。

（6）创建一个名为 applePicker 的类，该类继承自 fruitPicker 类。定义该类的__construct 魔术方法，使得该类被实例化的同时，其私有属性$fruitName 就被定义为"apple"。再按照类似的步骤分别创建 pearPicker 和 bananaPicker 类。

（7）分别输出苹果、梨和香蕉的数量，以及筐中所有水果的数量。

第 9 章　Web 编程基础

在之前的 8 章里，我们学习了如何搭建 PHP 开发环境、如何编写 PHP 脚本、如何使用条件和循环语句控制脚本的运行顺序以及如何使用函数、类和对象来复用脚本。在掌握了这些基础知识后，终于要开始接触 Web 编程了。

所谓的 Web 编程，其实就是通过编程实现基于 Web 浏览器的人机互动。这种人机互动要求人（也就是我们通常说的用户）输入一些信息，然后机（可以是与用户直接接触的计算机、该计算机上安装的浏览器，或者是远程的 PHP 服务器）根据用户输入的信息做出相应的反应。在这个过程里，信息的输入是通过 HTML 实现的，输入信息的处理则需要使用客户端脚本（如 JavaScript），或服务器端脚本（如 PHP）来实现。

那么这个人机互动的过程是如何实现的呢？我们可以倒回去看看图 1-2。在那幅图中可以知道用户是通过浏览器向服务器发出请求的，这些请求里可能会带有某些用户输入的信息。当服务器收到这些请求之后，会将其中用户输入的信息代入到 PHP 脚本中进行处理，然后将处理的结果以 HTML 文件的形式返回到浏览器，从而实现人机互动。

用户在通过浏览器向服务器发出请求和传递信息时，使用了统一资源定位符（URL）来确定访问对象并向访问对象传递数据。而当服务器返回信息后会将某些经常会使用到的信息以 cookies 的形式存储在本地缓存中以加快信息处理速度。为了保障人机互动整个过程的安全，我们还使用了会话（session）机制来避免这个过程受到恶意攻击。具体来说，每个用户对任意一个访问对象的每一次访问都可以被看做是一个会话。

在本章里，我们就来具体了解一下 URL、cookies 和 sessions。

9.1　使用 URL 传递数据

几乎所有的支持人机交互的网站都是使用 URL 和表单来传递数据的。现在以某搜索引擎为例，来看看如何使用 URL 传递数据。

首先在该搜索引擎的首页上输入“PHP”，然后单击“搜索”按钮。搜索结果如图 9-1 所示。

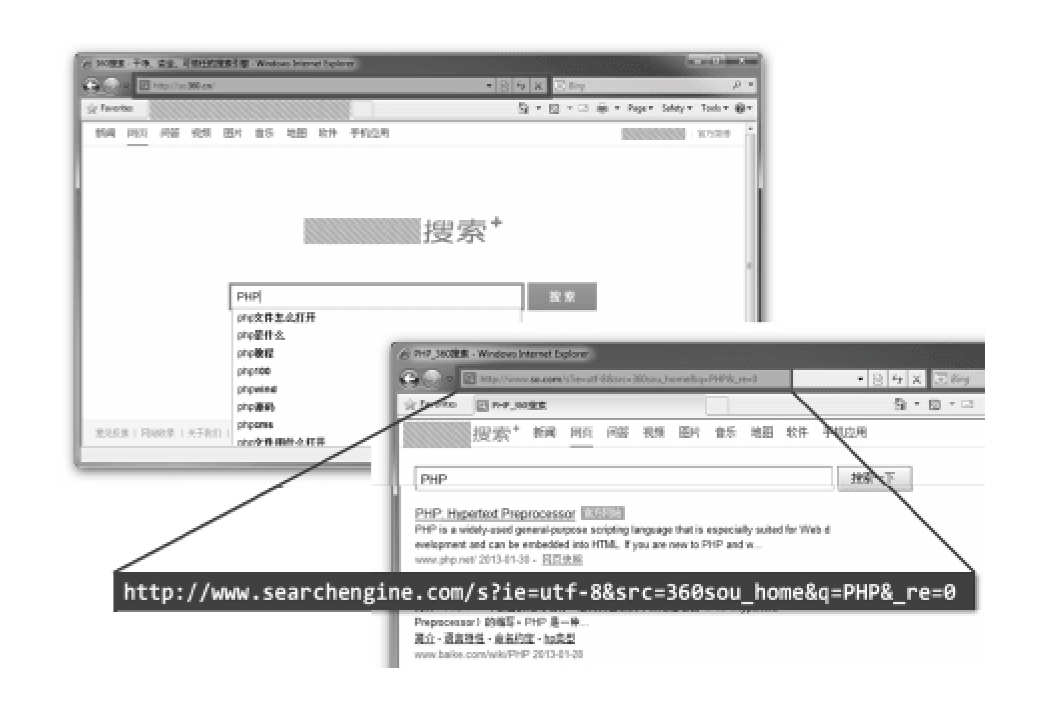


图 9-1　使用搜索引擎搜索 PHP 相关网页

在图 9-1 所示中，重点关注一下搜索结果页面的 URL（也就是我们常说的网址）是个什么样子。

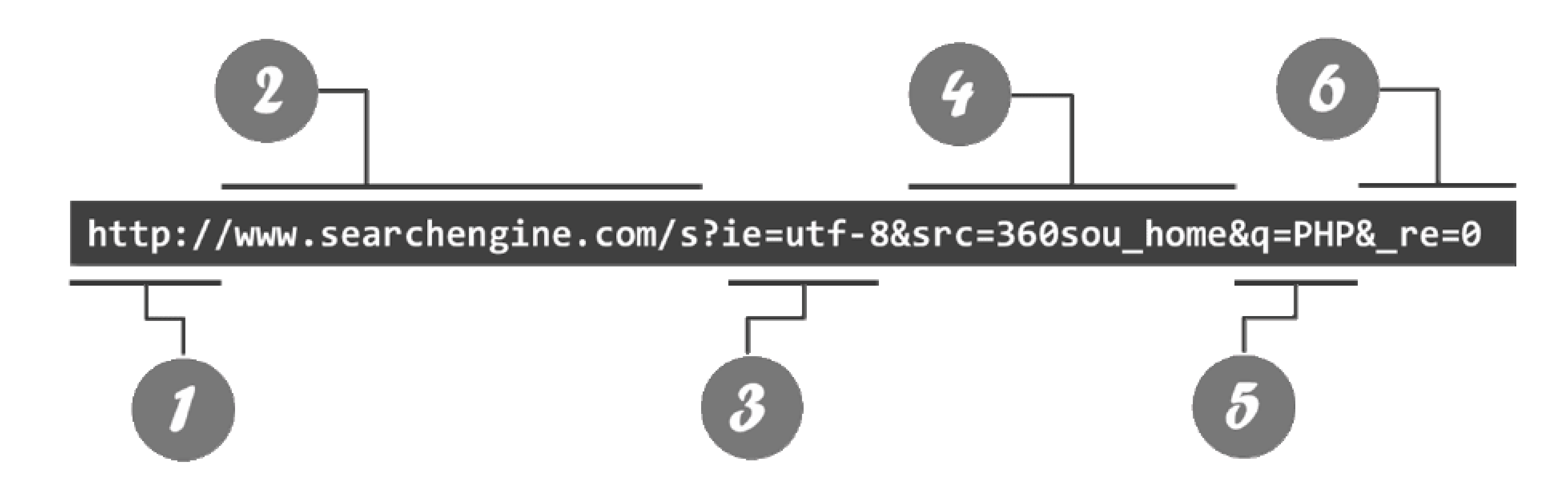


图 9-2　搜索结果页面的 URL

在上面这个网址中，我们可以看到有 6 个部分。在这 6 个部分中，除了“?”前面的文本传输协议名称、域名及文件名外，“?”后面的四个部分就是通过 URL 传递的参数字段：

- 字段 1 的参数名为 ie，参数值为 utf-8，用来表示浏览器和页面编码类型。
- 字段 2 的参数名为 src，参数值为 360sou_home，用来表示搜索来源页面。
- 字段 3 的参数名为 q，参数值为 PHP，用来表示搜索关键字。
- 字段 4 的参数名为_re，参数值为 0，具体代表的意义不甚了了。

值得注意的是，所有的参数字段都放在“?”后并使用“&”符号连接。

我们只是在搜索框里输入了一个简单的关键字，结果却产生了四个关联的参数字段。在这四个参数字段中，字段 1 应该是通过引用某系统常量得出的，字段 2 和字段 4 应该是由提交隐藏文本框传递的，而字段 3 就是我们在搜索页面上输入的内容。从这里可以看出，我们可以要求用户使用表单输入某些信息，然后将这些信息通过 URL 传递到其他页面。

在这一节里，我们就来详细了解一下如何实现这一过程。

9.1.1　收集用户信息

如果大家对 HTML 有一定的了解，应该知道表单是什么。例 9.1 就展示了一段 HTML 代码，我们一起来看看。

【例 9.1】 收集信息。

先新建一个 Ex9-01.html 文件，然后在文件中输入如下代码。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/
TR/html4/strict.dtd">
<meta content="text/html; charset=UTF-8" />
<html>
    <head>
        <title>A Form</title>
    </head>
    <body>
        <h2>Enter your name, please!</h2>
        <form action="Ex9-01.php" method="get">
            <h3>Name</h3>
            <input type="text" name="fullname" />
            <input type="submit" value="Submit" />
        </form>
```

```
    </body>
</html>
```

在这段 HTML 代码里，我们重点关注一下 form 标签里的 action 属性和 method 属性。action 属性的值是一个文件名，也就是说通过这个表单提交的数据会被传递到 Ex9-01.php 里进行处理；而 method 属性的值是 get，也就是说这些通过这个表单提交的数据会以明文方式传递到通过 action 属性指定的页面中。

现在，新建一个文件，将其命名为 Ex9-01.php。接着，运行 Ex9-01.html、在文本框中输入名字、单击“提交”按钮。发现 Ex9-01.html 消失了，取而代之的则是 Ex9-01.php，如图 9-3 所示。

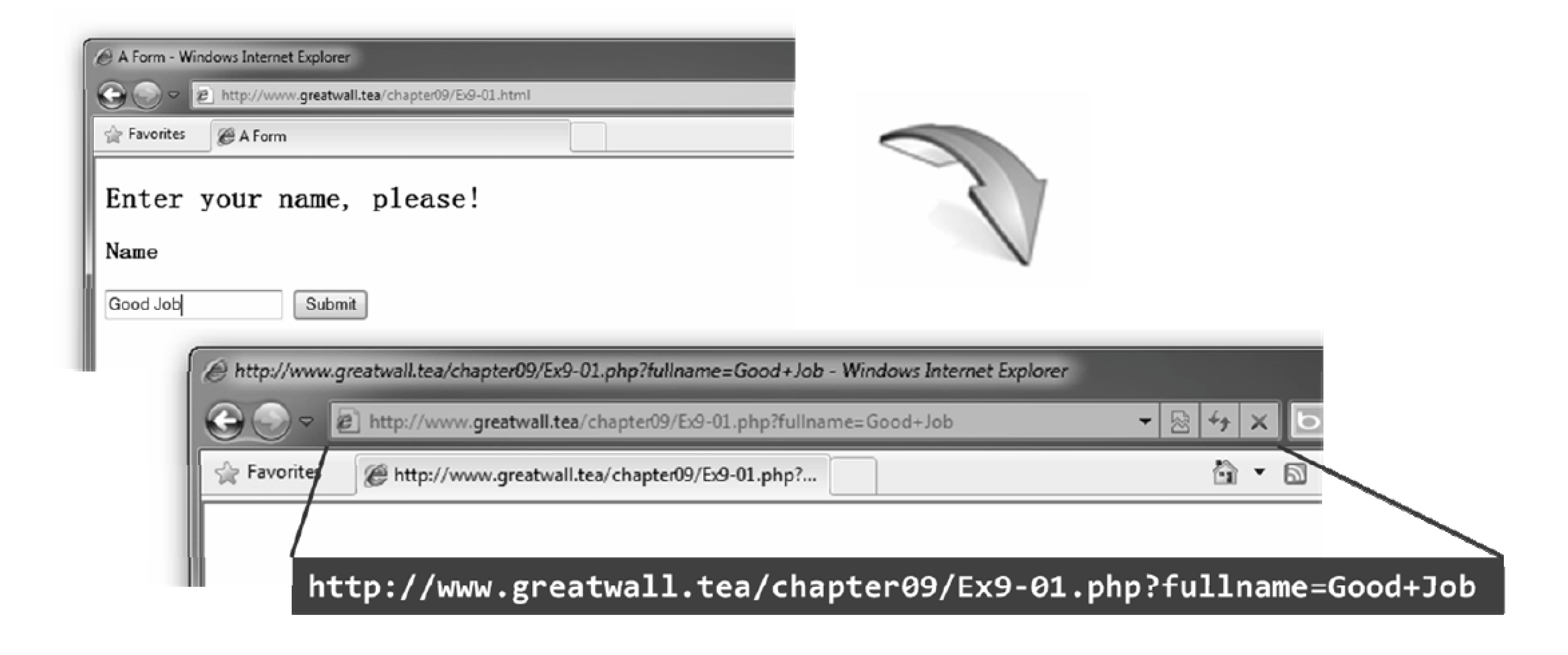


图 9-3　提交表单之后

但是，页面的网址后半部分居然带有参数字段。仔细打量一番，这个字段的参数名是“fullname”，而参数值则是“Good+Job”：前者是表单中文本框元素的 name 属性值，而后者则是在 Ex9-01.html 中输入的名字。

这样一来，数据就由 Ex9-01.html 传递到了 Ex9-01.php 中了。

需要说明的是，使用明文传递数据会比较危险，很容易造成信息泄露。为了防止出现这种情况，我们可以使用 post 方法来隐藏被传递的字段。不过这样一来，服务器的开销无形中就变大了，一旦访问流量过大，就很容易出现资源耗尽、服务器宕机的情况。其实我们可以使用其他的方式来对以明文方式传递的数据进行加密，从而在降低服务器开销的同时，也保护了数据的私密性。

9.1.2　接收信息数据

上一小节里，我们使用表单向一个页面传递了从用户处收集到的信息。在本小节里，就来看看如何接收这些数据。

【例 9.2】 接收数据的使用。

在 Ex9-01.php 中输入如下代码：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
```

```
<meta content=t"text/html; charset=UTF-8" http-equiv="Content-Type" />
<html>
    <head>
        <title>Welcome!</title>
    </head>
    <body>
        <h2>
            Welcome! <?php echo $_GET['fullname'];?>
        </h2>
    </body>
</html>
```

在这段代码中，使用了一段 PHP 脚本用来输出从 Ex9-01.html 的表单中收集到的数据。现在我们再次打开 Ex9-01.html 文件，输入一个名字，看看结果到底如何？



图 9-4　接收数据

按照图 9-4 所显示的结果，在 Ex9-01.html 中输入的名字成功地显示在了 Ex9-01.php 中。大家可以重点关注一下例 9.2 中加粗的那一段代码。在这段代码中使用了$_GET 数组，用来获取表单中指定元素的值。

PHP 一共为我们提供了三个预置数组，用来存储从表单中获取的数据。除了刚才使用的$_GET 数组之外，还有$_POST 数组和$_REQUEST 数组。$_GET 数组只有当表单传递数据的 method 设置为 get 时才会生效。类似地，$_POST 数组只有当表单传递数据的 method 设置为 post 时才会生效。而$_REQUEST 数组则对任意一种数据传递方式都有效。

为了加深印象，再来做一个比较复杂一点的。在例 9.3 所示中，Ex9-02.html 里有四个参数需要传递到 Ex9-02.php 中进行处理，它们分别是：姓名、性别、出生日期和婚姻状况。Ex9-02.php 页面在接收到 Ex9-02.html 传递过来的数据后，根据性别、出生日期和婚姻状态的不同，显示不同的问候语。

【例 9.3】　属于你的问候。

新建 Ex9-02.html 文件，在其中输入如下 HTML 代码：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<meta content="text/html; charset=UTF-8" http-equiv="Content-Type">
<html>
    <title>
        Please fill the form!
```

```
    </title>
    <body>
        <h2>
            Please fill the form!
        </h2>
        <form action="Ex9-02.php" method="get">
            <p>
                First Name 
                <input type="text" name="firstName" />
            </p>
            <p>
                Last Name 
                <input type="text" name="lastName" />
            </p>
            <p>
                Gender 
                <input type="radio" name="urGender" value="Male" checked="" />
                Male
                <input type="radio" name="urGender" value ="Female" />
                Female
            </p>
            <p>
                Birthday 
                <input type="text" name="urBirthday" value="1980-01-01" />
                  * Eg:1980-01-01
            </p>
            <p>
                Marital Status 
                <select name="urMarriage">
                    <option value="Married">Married</option>
                    <option value="Unmarried">Unmarried</option>
                </select>
            </p>
            <p>
                <input type="submit" value="Submit" />
                <input type="reset" value="Reset"/>
            </p>
        </form>
    </body>
</html>
```

接下来，再新建 Ex9-02.php 文件，用来接收和处理 Ex9-02.html 文件中传递来的数据。还记得我们应该如何接收由其他文件传递来的数据吗？

```
<?php
    //从 Ex9-02.html 中获取数据
    $firstName = $_REQUEST['firstName'];
    $lastName = $_REQUEST['lastName'];
    $gender = $_REQUEST['urGender'];
    $birthday = $_REQUEST['urBirthday'];
    $marriage = $_REQUEST['urMarriage'];

    //计算用户的年龄
    function getAge($birthday) {
        //obtain the year in $birthday
        $birthYear = substr($birthday, 0, 4);

        //obtain the current year
        $currentYear = date("Y", time());
```

```

        //calculate the age
        return $currentYear - $birthYear;
    }

    //给用户定称谓
    function getTitle($gender, $marriage) {
        if (substr($gender, 0, 1) == "M") {
            return "Mr.";
        } else {
            if (substr($marriage, 0, 1) == "M") {
                return "Mrs.";
            } else {
                return "Miss";
            }
        }
    }
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<meta content="text/html; charset=UTF-8" http-equiv="Content-Type">
<html>
    <title>Welcome</title>
    <body>
        <h1>
            Welcome, <?php echo getTitle($gender, $marriage)."
            ".$_REQUEST['lastName'] ?>!
        </h1>
        <p>You are <?php echo getAge($birthday) ?> years old now!</p>
    </body>
</html>
```

这段脚本首先定义了五个变量用来获取从 Ex9-02.html 中传递过来的数据，然后定义了两个函数 getAge()和 getTitle()来确定用户的年龄和称谓。现在运行 Ex9-02.html，并输入如图 9-5 所示的内容，然后单击“Submit”按钮。在页面跳转到 Ex9-02.php 后，我们在 Ex9-02.html 中的表单里填写的所有内容都可以在 Ex9-02.php 的 URL 中找到。



图 9-5　属于你的问候

现在，试着修改一下 Ex9-02.html 表单里各项目的值，看看结果有什么变化吧？

9.1.3 检测接收到的数据

假如在 Ex9-02.html 中，输入了如图 9-6 所示的内容。那么 Ex9-02.php 中显示的问候可能会有些不妥。

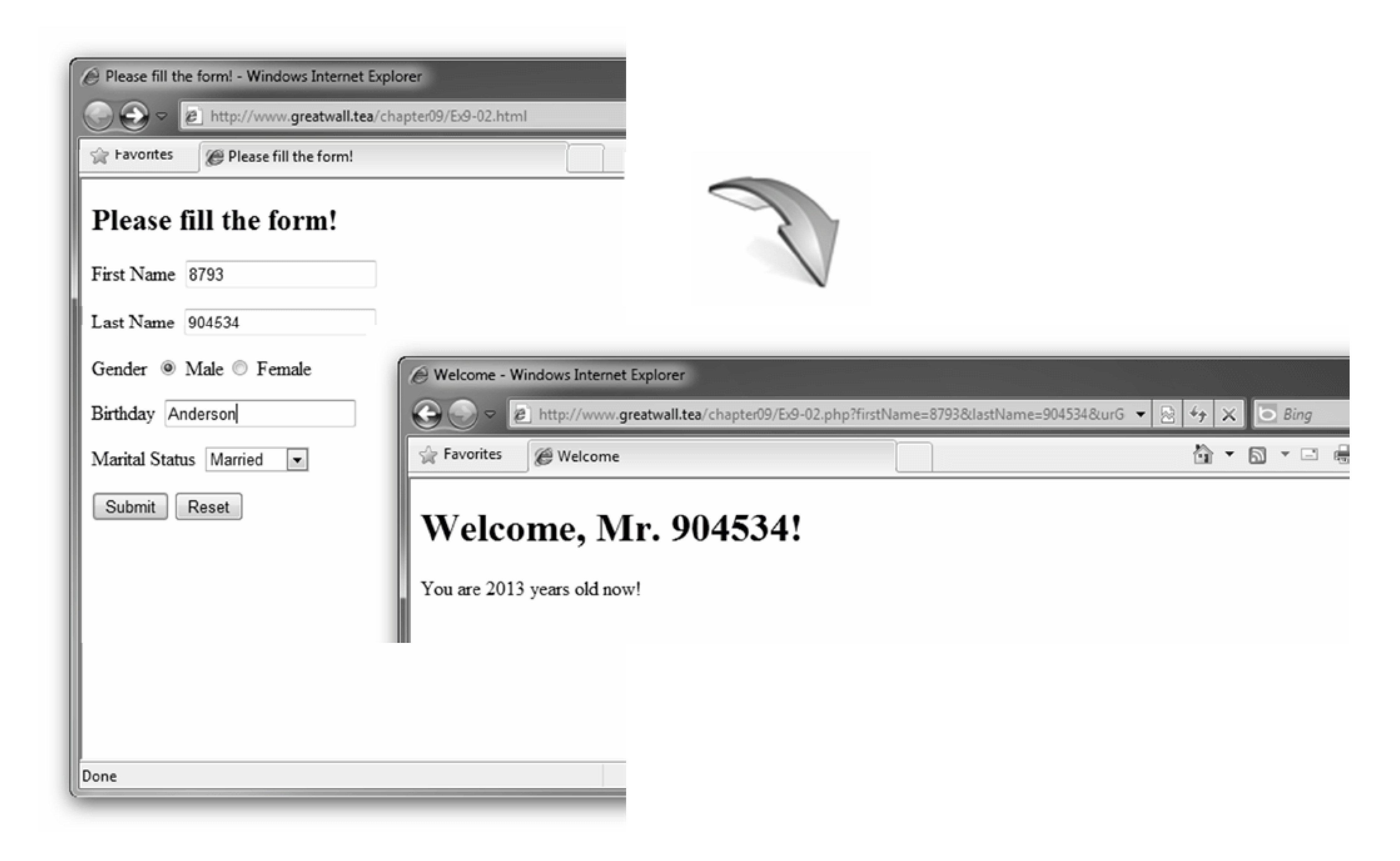


图 9-6 不妥的问候

如图 9-6 所示，我们在 First Name 和 Last Name 两个栏目里输入了数字，而在应该填写日期的生日一栏里却输入了人名。这样一来，虽然系统没有报错，但是出现的结果却十分的怪异。

为了避免出现这种尴尬的情况，我们需要对从其他页面接收到的数据进行检测。如果接收到的某些数据不符合要求，就应该提示用户进行必要的修改。

现在就着手对 Ex9-02.php 进行改造：为其添加数据检测的功能。

【例 9.4】 参数检测。

首先，将 Ex9-02.html 复制一份，并将其重命名为 Ex9-03.html。然后将文件中的第 12 行由：

```
12          <form action="Ex9-02.php" method="get">
```

改成：

```
12          <form action="Ex9-03.php" method="get">
```

接着，新建 Ex9-03.php 文件，并在文件中输入：

```
<?php
    //获取从 Ex9-03.html 传递来的数据
    $firstName = $_REQUEST['firstName'];
```

```
        $lastName = $_REQUEST['lastName'];
        $gender = $_REQUEST['urGender'];
        $birthday = $_REQUEST['urBirthday'];
        $marriage = $_REQUEST['urMarriage'];
        $errMsgHead = "Attention!";
        $errMsgCont = array('03256' => "Invalid name.
                                A name must start with a uppercase letter
                                and contain only letters.",
                        '04218' => "Invalid date.
                                The format of a valid birthday is similar to
                                1980-01-12.");

        $errCount = 0;
        $errMsg = "";

        /用于检测$firstName 和$lastName 是否正确的函数
        function validateName($name) {
           return preg_match("/^[A-Z][a-z]*$/", $name);
        }

        //用于检测$birthday 的值是否正确函数
        function validateBirth($birthday) {
           return preg_match("/^[0-9]{4}(\-(0[1-9]|1[0-2]))(\-(0[1-9]|
           1[0-9]|2[0-9]|3[0-1]))$/", $birthday);
        }

        //开始检测相关参数
        $is_firstName = validateName($firstName);
        $is_lastName = validateName($lastName);
        $is_birthday = validateBirth($birthday);

        //计算用户的年龄
        function getAge($birthday) {
           //获取$birthday 中的年份
           $birthYear = substr($birthday, 0, 4);

           //获取当前年份
           $currentYear = date("Y", time());

           //返回用户年龄
           return $currentYear - $birthYear;
       }

       //获取用户称谓
       function getTitle($gender, $marriage) {
           if (substr($gender, 0, 1) == "M") {
               return "Mr.";
           } else {
               if (substr($marriage, 0, 1) == "M") {
                   return "Mrs.";
               } else {
                   return "Miss";
               }
           }
       }
    ?>
    <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">
    <meta content="text/html; charset=UTF-8" http-equiv="Content-Type">
```

```
62  <html>
63      <head>
64          <?php
65              if ($is_birthday == FALSE ||
66                $is_firstName == FALSE ||
67                $is_lastName == FALSE){
68                  echo $errMsgHead;
69              } else {
70                  echo "Welcome";
71              }
72          ?>
73      </head>
74      <body>
75          <?php
76              if ($is_birthday == FALSE ||
77                $is_firstName == FALSE ||
78                $is_lastName == FALSE){
79          ?>
80          Before you continue, ratify the following errors:
81          <ul>
82          <?php
83                  if ($is_firstName == FALSE || $is_lastName == FALSE)
84                      echo "<li>".$errMsgCont['03256']."</li>";
85                  if ($is_birthday == FALSE)
86                      echo "<li>".$errMsgCont['04218']."</li>";
87          ?>
88          </ul>
89          <?php
90              } else {
91          ?>
92          <h1>Welcome, <?php echo getTitle($gender, $marriage)."
            ".$_REQUEST['lastName'] ?>!</h1>
93          <p>You are <?php echo getAge($birthday) ?> years old now!</p>

94          <?php
95              }
96          ?>
97          <div align="center">
98              <a href="Ex9-03.html">Return</a>
99          </div>
100     </body>
101 </html>
```

在编写完这两个文件后，运行 Ex9-03.html，然后在表单中输入如图 9-7 所示的内容，单击“Submit”按钮。期待中的欢迎页面并未出现，取而代之的却是两条错误提示。这就意味着，我们在 Ex9-03.php 中编写的参数检测机制发挥了作用。

在实现参数检测机制的过程中，我们使用了许多之前学过的内容，包括数组、正则表达式和自定义函数等。在这里一起来详细地讲解这段代码。

具体来说，在上面这段脚本里完成了如下几项任务。

（1）接收由 Ex9-03.html 传递来的数据

在脚本的 2～7 行使用$_REQUEST[]获取了由 Ex9-03.html 传递来的数据，并将它们赋值给若干变量备用。

（2）定义出错信息

由于在脚本中添加了参数检测机制，因此需要在用户输入的信息不符合要求时给予用户相应的提示。在脚本的 8～14 行定义了一个变量$errMsgHead 和一个数组$errMsgCont

用来存放提示信息。

（3）定义用于检测关键参数的函数，并检查关键参数

在脚本的 16～27 行定义了 validateName()和 validateBirth()两个函数来检查用户输入的姓名和出生日期是否符合格式要求。在这两个函数中，我们使用了正则表达式和 preg_match()函数。然后在 29～32 行里，对用户输入的 First Name、Last Name 和 Birthday 进行检查，并将检查结果放入$is_firstname、$is_lastname 和$is_birthday 三个变量中。

（4）计算用户年龄，获取用户称谓

在脚本的 34～57 行定义了 getAge()和 getTitle()两个函数来根据用户输入的出生日期来计算用户年龄，根据用户选择的性别和婚姻状况来确定用户的称谓。

（5）确定 Ex9-03.php 页面的标题

在脚本的 64～72 行使用 if…else 语句判断了$is_firstname、$is_lastname 和$is_birthday 三个变量的值。只要这三个变量中的值任意一个为 FALSE，则页面标题为$errMsgHead 的值，否则页面标题为“Welcome”。

（6）确定 Ex9-03.php 页面内容

在脚本的 74～100 行使用了 if…else 语句的嵌套，以便根据用户输入的内容动态显示不同的内容。若用户输入的 First Name 或 Last Name 不符合格式要求，则 Ex9-03.php 的页面标题为“Attention!”，页面内容包含了一条关于“Invalid name.”的提示。若用户输入的 Birthday 不符合格式要求，Ex9-03.php 的页面标题为“Attention!”，页面内容则包含了一条关于“Invalid date.”的提示。若这三个关键参数的格式均不正确，Ex9-03 的页面标题仍然为“Attention!”，页面内容则包含了上述两条错误提示。只有当这三个关键参数的格式均正确时，Ex9-03 的标题和内容才会如图 9-5 所示。

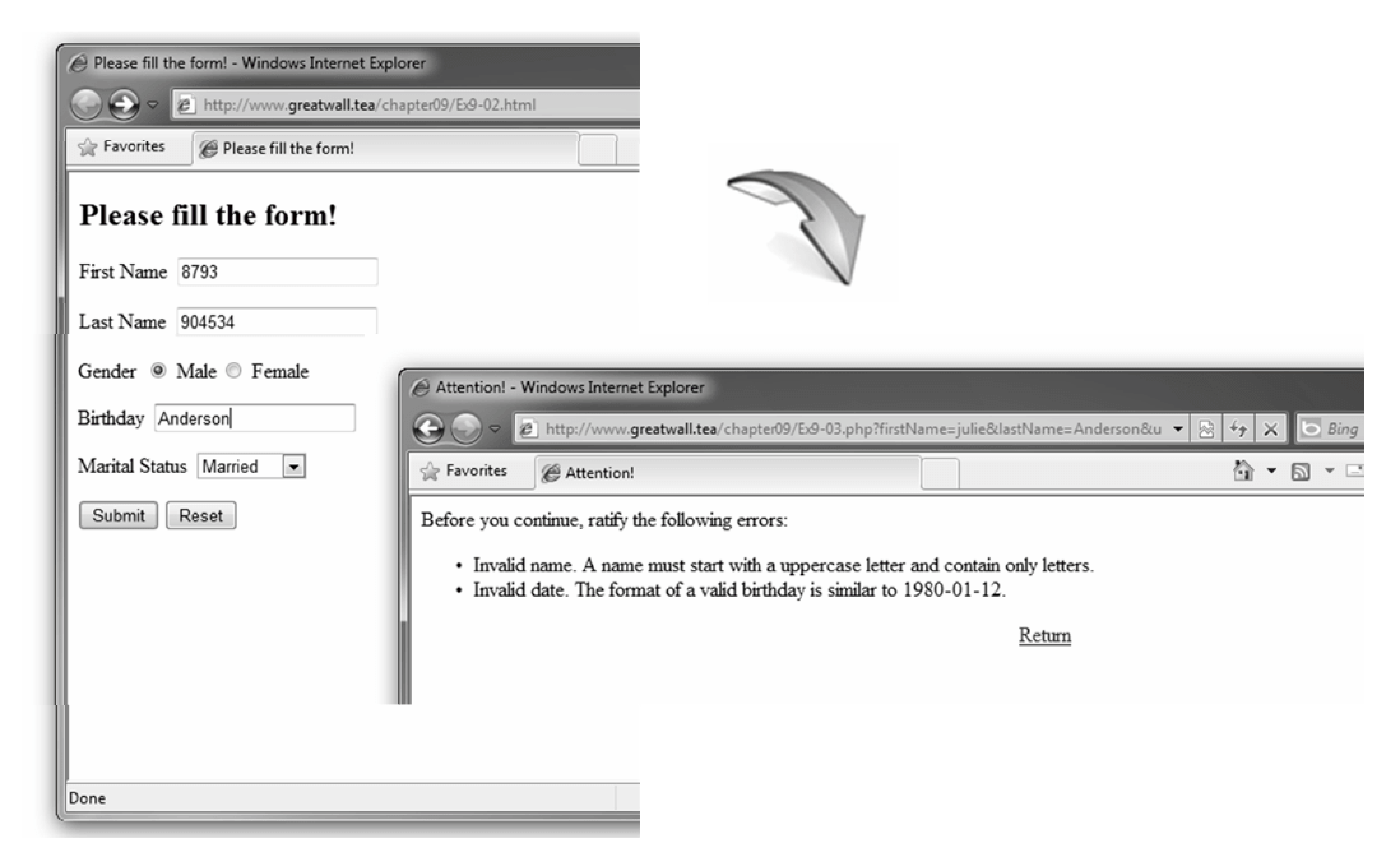


图 9-7　参数检测

总结一下，在上面的这段脚本里，接收了由其他页面传递来的数据、检查数据的格式、根据用户输入的数据进行用户年龄和称谓的计算，最后输出计算结果。若用户输入的数据

不符合要求，输出提示信息帮助用户改正。

9.2　使用 Cookie 缓存数据

大家上网的时间一长，就会产生很多的隐私数据。这些隐私数据除了占据了本地存储空间之外，还有可能造成隐私泄露。因此，大多数的浏览器都提供了清理浏览数据的功能。而 Cookie 就是隐私数据的一种。图 9-8 展示了由 Google 开发的开源浏览器 Chrome 中清理浏览数据的界面。

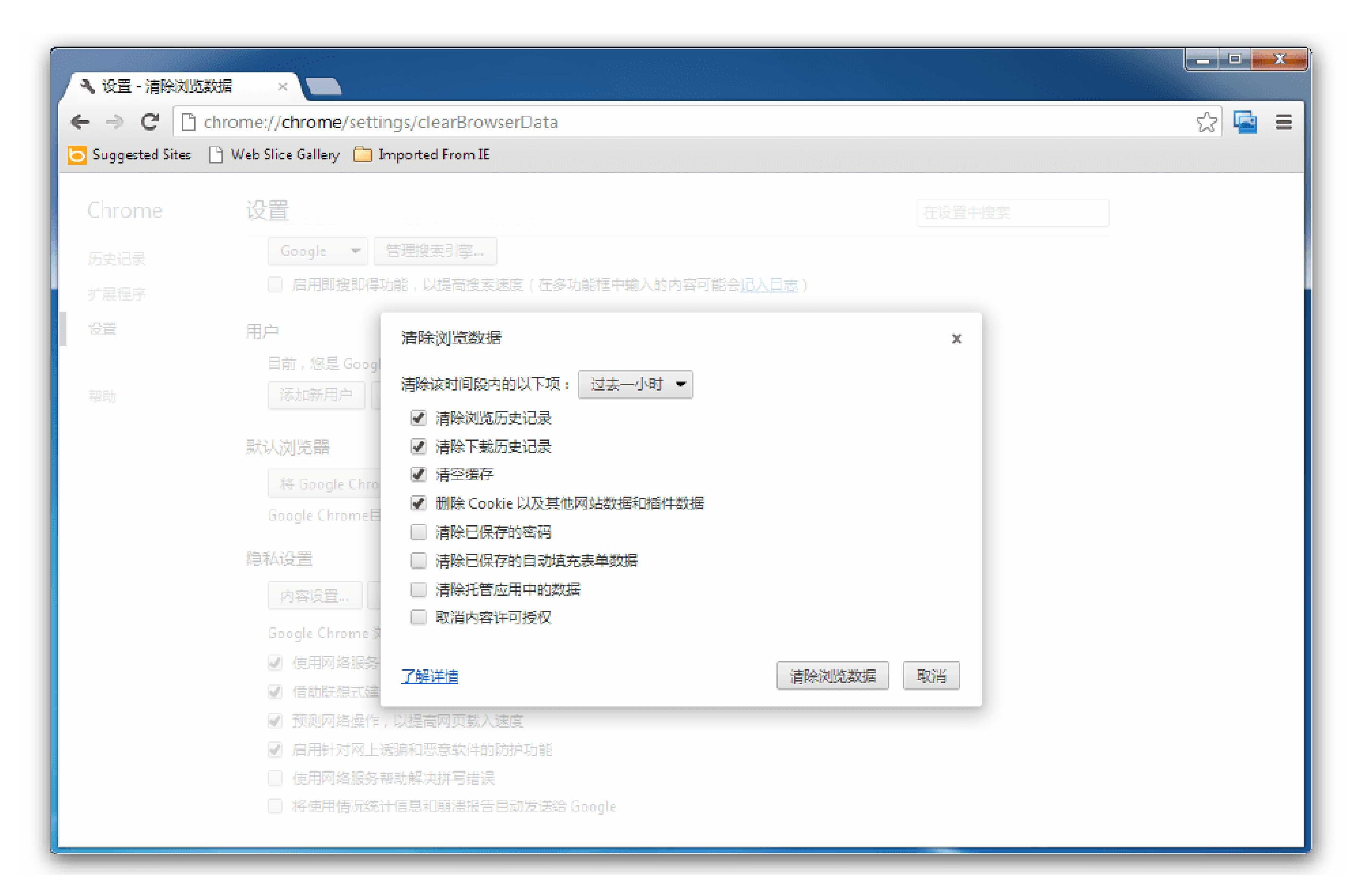


图 9-8　Google Chrome 浏览器清除浏览数据

其中，第四项即为“删除 Cookie 以及其他网站数据和插件数据”。那么，到底什么是 Cookie 呢？按照维基百科的解释，Cookie 是某些网站为了辨识用户身份而存储在本地终端上的经过加密的数据。

这说明，通过 Cookie 缓存的数据有可能被用户自行清空而无法获取。因此，指望通过这种方法将数据缓存到本地基本上是不太现实的事情。即使如此，我们还是有必要了解一下，如何在本地缓存少量数据，以达到提升用户浏览体验的目的。

9.2.1　使用 Cookie 存取数据

在 PHP 中，可以使用 setcookie 方法在用户本地存储 Cookie 信息。具体用法如下：

```
setcookie("variable","value");
```

存储在用户本地的 Cookie 数据与通过 URL 传递的数据的格式一样，都是键值对。也就是说上面这条语句中的 variable 是“键”，而 value 则是“值”。需要注意的是，在“键”的前面是不需要使用“$”符号的。例如，我们可以使用下面这条语句，把“prov=Beijing”

这条信息存储在本地。

```
setcookie("prov","Beijing");
```

当用这条语句把信息存储在本地后，可以使用 PHP 内置的数组$_COOKIE 来提取这些信息。比如，可以使用下面这条语句来显示我们刚才存储在用户本地的信息：

```
echo "You are from ".$_COOKIE['prov'].".";
```

这条语句输出的结果应该是：

```
You are from Beijing.
```

需要注意的是，在设置$_COOKIE 数组的页面中，系统无法读取 Cookie 数据。只有当用户离开该页面前往其他页面时，才可以使用$_COOKIE 数组读取已存储的 Cookie 信息。

通常情况下，我们会把用户名等一些涉及用户隐私的信息通过 Cookie 存储在用户本地，从而方便用户身份的认证。但是这也给不怀好意的人有机可乘。为了尽可能的避免这种情况，可以给 Cookie 信息设置有效期。

```
setcookie("variable", "value", expiretime);
```

在这条语句中，前两个参数与之前无异；最后一个参数 expiretime 指的是该 Cookie 过期的时间点。我们可以使用之前学过的 time()或 mktime()函数来指定。

- time()函数以时间戳的形式返回当前的系统时间，单位为秒可以在其后加上一定时间的秒数来设定一个未来的时间点，比如：

```
setcookie("prov", "Beijing", time() + 3600);     //Cookies 将在 1 小时后过期
setcookie("Name", $name, time() + (3*86400));  //Cookies 将在 3 天后过期
```

- mktime()函数则可以直接指定一个时间点，比如：

```
setcookie("prov", "Beijing", mktime(3, 0, 0, 4, 1, 2013));//Cookies 将在
2013 年 4 月 1 日早上 3：00 过期
```

需要注意的是 mktime()函数的参数分别为：时、分、秒、月、日、年。

9.2.2　销毁 Cookie 数据

在为一条 Cookie 数据设置了过期时间点后，时间一到，数据就会自动失效，我们也就无法通过$_COOKIE 数组调取该条数据了。当然，如果你想在该条数据到期前手动销毁它，可以使用 setcookie()方法重新定义该 Cookie 的键对值，将其值设置为空即可，比如：

```
setcookie("prov", "");
```

9.2.3　关于 Cookie 的后话

在知道如何使用 Cookie 存取数据和销毁数据后。还是想跟大家讲讲何时使用 Cookie。

由于 Cookie 数据存储在用户本地，越来越多的用户对这种机制心存芥蒂。许多的用户干脆在本地禁用了 Cookie 机制，从而禁止所有的网站于用户浏览时在本地存取数据。因此，不要过度的依赖 Cookie 机制在用户本地存储大量的信息。一旦用户禁用了 Cookie 功能，所有的数据以及与这些数据相关的功能都会失效。

另外，还需要注意的是，setcookie()方法有其局限性。不能在 PHP 脚本已经向页面输出信息后再定义 Cookie 数据，否则 setcookie()无效。

9.3　使用 Session 保障数据安全

与 Cookie 数据存储在用户本地不同，Session 数据存储在服务器上。一个 Session 是指某用户在某网站上逗留的整段时间，这段时间可以称其为一个会话期间，或简称其为一条会话。在用户开始进入一个网站到离开该网站的这段时间内，我们可以使用 Session 机制使该网站的所有页面共享某些数据，从而提升用户的浏览体验。

9.3.1　PHP Session 工作机制

我们可以使用 PHP 脚本创建和存储 Session 变量。在创建了一个 Session 后，所有 Session 变量在用户一次会话期间里访问的所有页面都有效。其工作机制如图 9-9 所示。

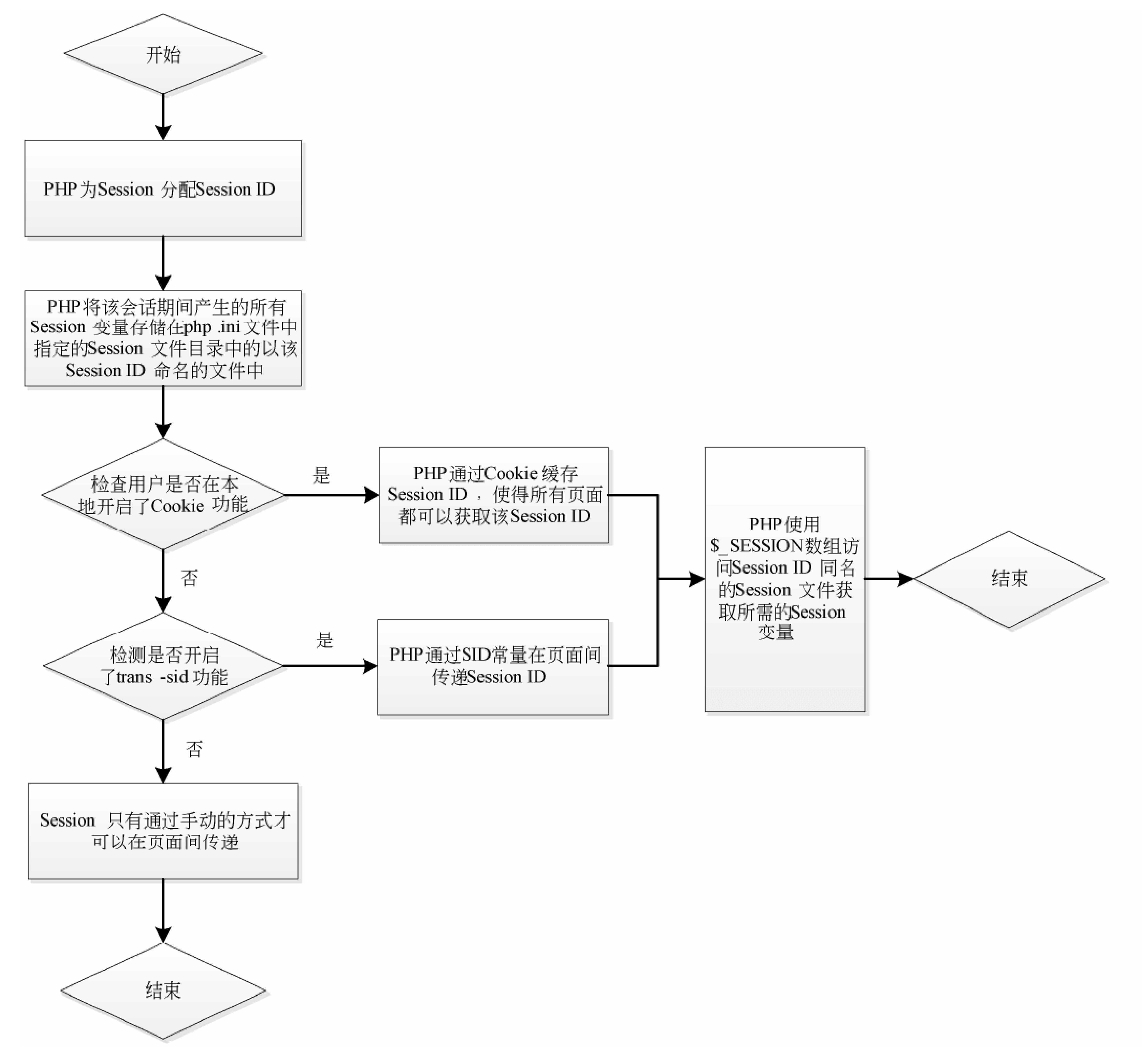


图 9-9　PHP Session 工作机制流程图

通过上面这张流程图，可以清楚地知道 PHP 的会话机制。如果大家的网站架设在共享服务器上，且访问该网站的用户没有开通 Cookie 功能，（若非通过手动的方式在所有的页面中传递 Session ID）整个网站的 Session ID 就有可能失效。

现在详细地讲讲如何创建 Session、传递 Session ID、存取 Session 变量及销毁 Session 的内容。

9.3.2　创建及销毁 Session

只有在每个页面上都开启了 Session 功能，Session 变量才可以在页面间传递。开启 Session 功能的方法如下：

```
session_start();
```

这条语句会检查现有的 Session ID。如果检查到一个 Session 的存在，则开始创建 Session 变量；若未检查到任何 Session，则开始创建新的 Session。

需要注意的是，session_start()方法与 set_cookie()的方法一样，都必须在 PHP 向客户端页面输出任何信息之前执行，否则 Session 创建失败。因此，最好的方法就是在每个页面的开始就执行 session_start()和 set_cookie()方法。

如果觉得这种办法有些麻烦，而你又可以修改服务器上的 php.ini 文件。那么在该文件中查找 session_autostart 变量，并将它的值修改成 1。这样，每当用户访问服务器上的任意页面时，都会自动执行 session_start()方法。

另外，也可以在用户可以访问某些页面之前对它们进行身份验证：只有通过身份验证的用户才允许其访问这些页面。那么就需要用户输入用户名和密码等元素来登录网站。当用户离开网站时，也需要其退出。关于登录，我们可以使用 session_start()方法，而对于退出，可以使用下面这条语句：

```
session_destroy()
```

这条语句执行后，当前 Session 文件内的所有变量都被重置。PHP 也不再向下一个页面传递 Session ID。但是执行该语句的页面不会受此影响。如果你需要将当前页面中的所有 Session 变量都失效，则需要使用类似如下的语句：

```
unset($variableName1, [$variableName2],…)
```

9.3.3　使用 Session 变量

在 9.3.1 小节中，我们提到了 Session 变量这个词。所谓的 Session 变量指的是存储在 Session 文件中，供后一次会话期间的所有文件调用的变量。若需要在后续页面中调用某些在当前页面中设置的变量，可以在当前页面中将这些变量以键值对的方式存入$_SESSION 数组中，如：

```
$_SESSION['varname'] = "John Smith";
```

在同一会话期间的后续页面中，可以像调用普通数组一样调用这个数组变量。

如果需要销毁该 Session 变量，需要使用 unset()方法，如：

```
unset($_SESSION['varname']);
```

在例 9.5 中将定义两个页面 Ex9-04a.php 和 Ex9-04b.php 来展示如何使用 Session 在两个页面间传递数据。

【例 9.5】 使用 Session 在页面间传递数据。

```
<?php
    session_start();
    $_SESSION["session_var"] = "session testing";
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<meta content = "text/html; charset=UTF-8" http-equiv="Content-Type"/>
<html>
    <head>
        <title>
            Session Test Page 1
        </title>
        <body>
            <p>This is a test for data transmission through sessions</p>
            <form action="Ex9-04b.php" method = "post">
                <input type="text" name="form_var" value="form testing"/>
                <input type="submit" value="Go to Next Page" />
            </form>
        </body>
    </head>
</html>
```

上面这一段脚本使用 session_start()方法开启了 Session 功能，接着，定义了一个 Session 变量$_SESSION['session_var']，其值为 session testing。然后，用 HTML 定义了一张表单，定义了 Ex9-04b.php 为处理该表单的动作，并规定使用 POST 方法传递数据。表单中有一个文本框，名为 form_var，值为 form testing。

现在来写 Ex9-04b.php 文件。

```
<?php
   session_start();
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<meta content="text/html; charset=UTF-8" http-equiv="Content-Type">
<html>
    <head>
        <title>
            Session Test Page 2
        </title>
    </head>
    <body>
        <?php
            $session_var = $_SESSION['session_var'];
            $form_var = $_POST['form_var'];
            echo "session_var is $session_var.<br>";
            echo "form_var is $form_var.";
        ?>
    </body>
</html>
```

在如 Ex9-04b.php 所示的这段脚本中，我们在页面的开头也使用了 session_start()方法开启了 Session 功能。接着，在 HTML 的 body 标签下，定义了两个普通变量$session_var

和 $form_var 用来存储会话变量和由表单传递过来的变量，它们的值分别为 $_SESSION['session_var']和$_POST['form_var']。

接着运行 Ex9-04a.php，然后单击“Go to Next Page”按钮，会发现页面显示了如下的两段话：

```
session var is session testing.
form_var is form testing.
```

在例 9.5 所示中，我们默认用户开启了 Cookies 功能，以便 PHP 在页面间传递 Session ID。按照图 9-9 所示的流程图来看，如果用户关闭了浏览器的 Cookies 功能。系统会检测服务器的 PHP 引擎上是否开启了 trans_id 功能。若已开启，PHP 引擎会做如下的工作：

- 设定一个常量，其名为 SID，值为一个形如 PHPSESSID=*longstringofnumber* 的键值对。
- 然后在一个用户会话期间，PHP 引擎会主动将该常量的值，也就是当前 Session 的 ID 传递到用户访问的每个页面。

当然，上面这个过程对用户来说是透明的。也就是说，用户不会感知到使用 Cookies 传递 Session ID 和使用 SID 常量配合 trans_id 传递 Session ID 的差别。但是，此时 Session ID 是以明文的方式附加在 URL 后面的。任何人都可以看到该 Session ID，容易造成安全问题。若用户在浏览器中收藏的页面 URL 中带有 Session ID，那么当用户单击收藏夹中收藏的地址访问该页面时，新的 Session ID 和旧的 Session ID 会发生冲突，进而引发一系列问题。也正是由于该原因，PHP 引擎中的 trans_id 功能默认是禁用的。

那么如果用户既没有开启 Cookies，服务器上也没有开启 trans_id 功能，我们就需要在页面间手动传递 Session ID 了。

在手动传递 Session ID 时，还是会使用 SID 常量。我们需要将其手动附加到需要访问页面的地址后面，如：

```
<a href = "nextpage.php?<?php echo SID?>" >Next Page</a>
```

假设当前会话的 Session ID 为 877c22163d8df9deb342c7333cfe38a7，那么常量 SID 的值就是 PHPSESSID=877c22163d8df9deb342c7333cfe38a7，上面这条语句指向的页面地址就应该是 nextpage.php? PHPSESSIONID=877c22163d8df9deb342c7333cfe38a7。

这样一来，Session ID 还是以明文的方式在页面间传递，会出现很多的问题。为了避免这种情况，我们也可以通过表单的隐藏文本框来帮助我们传递 Session ID，如：

```
<?php
    $PHPSESSID = session_id();
    echo "<form action='nextpage.php' method='post'>
            <input type='hidden' name='PHPSESSID' value='$PHPSESSID'>
            <input type='submit' value='Next Page'>
         </form>";
?>
```

上面这条语句使用了 session_id()函数，获取当前 Session 的 ID，然后定义了一张表单，并将获取到的 Session ID 填入了表单中的隐藏文本框内。当用户单击“Next Page”按钮时，Session ID 就会传递到下一个页面中。读者也可以使用 JavaScript 脚本将一个超链接转换为提交按钮实现通过单击链接提交 Session ID 的功能。

9.4　使用表单上传文件

我们除了可以在页面间传递数据外，还可以向服务器上传文件。例如，在许多的论坛里，如果用户需要上传文件的话，就需要我们提供一个上传文件的接口。

9.4.1　使用表单上传文件

首先，还是依葫芦画瓢，先看看其他的网站上是如何做文件上传这个功能的，如图 9-10 所示。

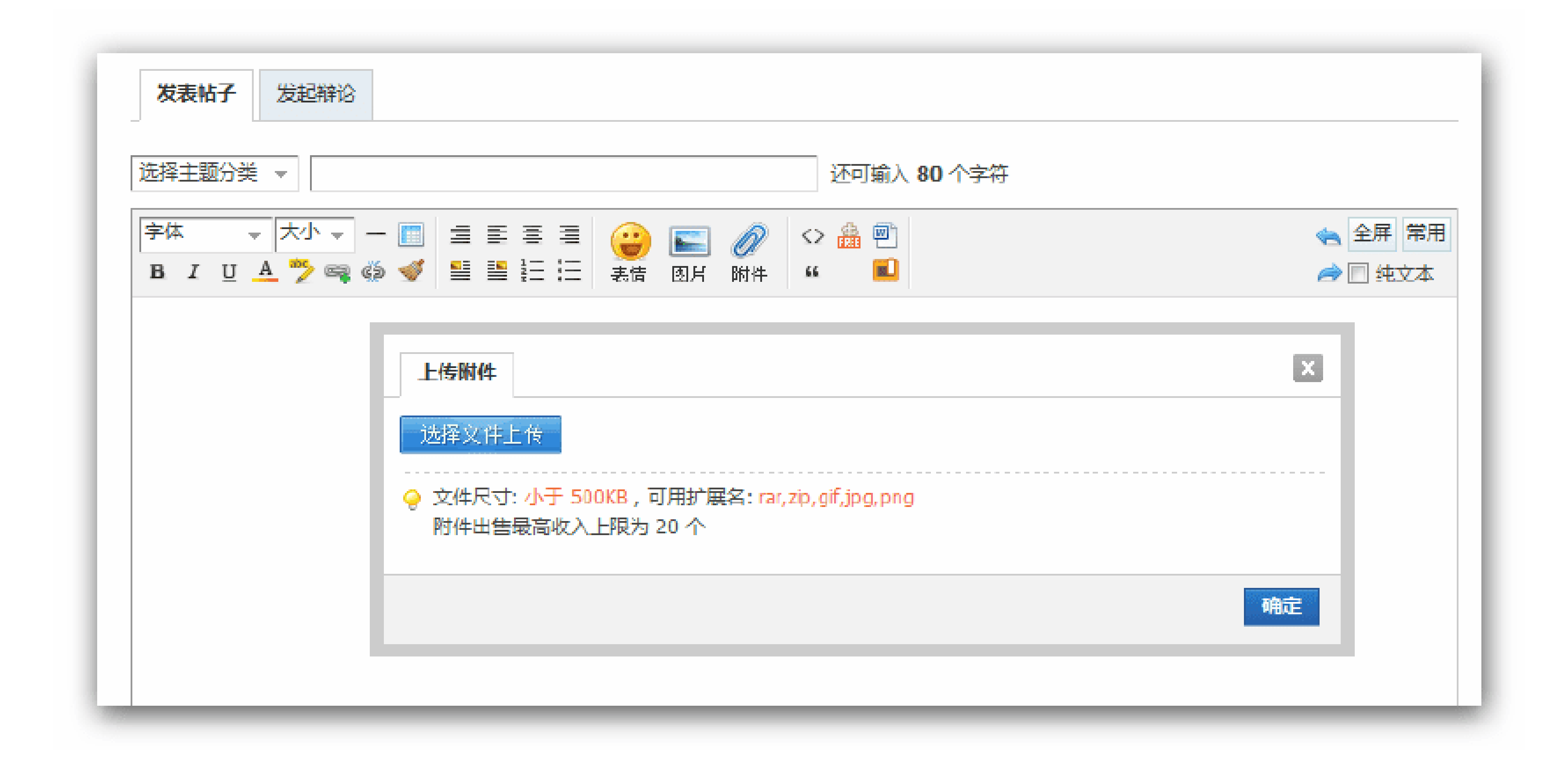


图 9-10　上传附件

在图 9-10 所示中可以看到，该网站上的文件上传组件是由两个按钮构成的，一个是"选择上传文件"按钮，另一个是"确定"按钮。在 HTML 中，原生的上传组件是通过类型为"file"的表单项定义的，具体如例 9.6 所示。

【例 9.6】 文件上传表单。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<meta content="text/html; charset=UTF-8" http-equiv="Content Type">
<html>
    <head>
        <title>Upload a file</title>
    </head>
    <body>
        <p>You can click the button to select the file you want to
        upload:</p>
        <form enctype="multipart/form-data" action="Ex9-05b.php"
        method="post">
            <input type="hidden" name="MAX FILE SIZE" value="30000">
            <input type="file" name="user file">
            <br>
            <input type="submit" value="Upload File">
        </form>
    </body>
</html>
```

上面这段 HTML 代码运行后，显示的内容如图 9-11 所示。

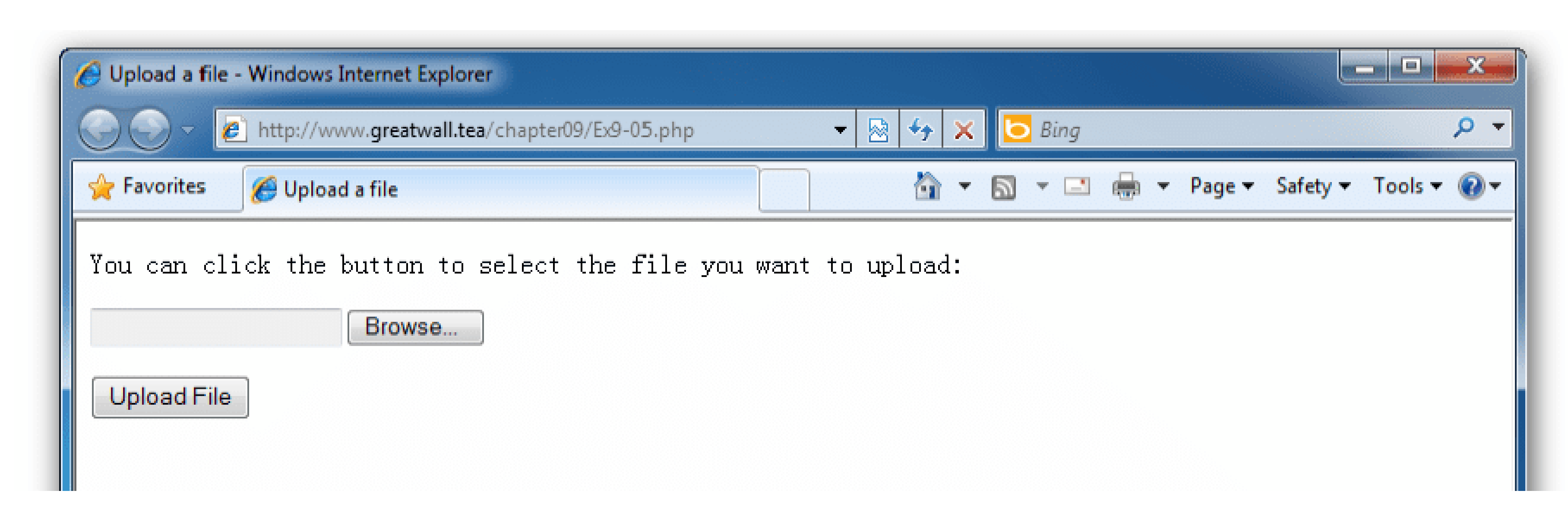


图 9-11　HTML 原生的文件上传组件

通过图 9-11 所示可以看到，HTML 原生的上传组件是由一个文件框和一个“Browse…”按钮构成的。图 9-11 所示中的两个按钮“Browse…”和“Upload File”就相当于图 9-10 所示中的“选择上传文件”和“确定”按钮。

关于例 9.6 中的脚本，还有两点需要说明：

- ❑ 如果需要通过表单上传文件，那么表单的“enctype”属性必须设置为“multipart/form-data”，否则文件上传后可能会出现错误。
- ❑ 另外，表单中使用了一个名为“MAX_FILE_SIZE”的隐藏文本框，用于指定用户可以上传文件体积的最大值。在设置这个值的时候，需要考虑 PHP 引擎的配置文件 php.ini 中两个变量的取值问题。一是“upload_max_filesize”，另一个是“post_max_size”。前者是允许上传的最大文件大小，后者是通过 POST 方法允许上传的最大文件大小。注意，如果使用 post 方法上传文件（我们一般都这么做），那么 upload_max_filesize 的大小不能大于 poast_max_size，否则一定会出现问题。如果你从来没有修改过 php.ini 文件中这两个参数的值，前者的默认值为 2 MB，后者的默认值为 8 MB。

当用户使用浏览器打开 Ex9-5.php 文件、选中某个文件并单击“Upload File”按钮后，用户选中的文件被上传到一个用于存放临时文件的文件夹内。我们需要编写脚本将其复制到某个可永久存放文件的文件夹内，以防文件在脚本执行结束后被删除。如果不知道你的服务器使用的临时文件夹，可以使用 phpinfo()方法查看。

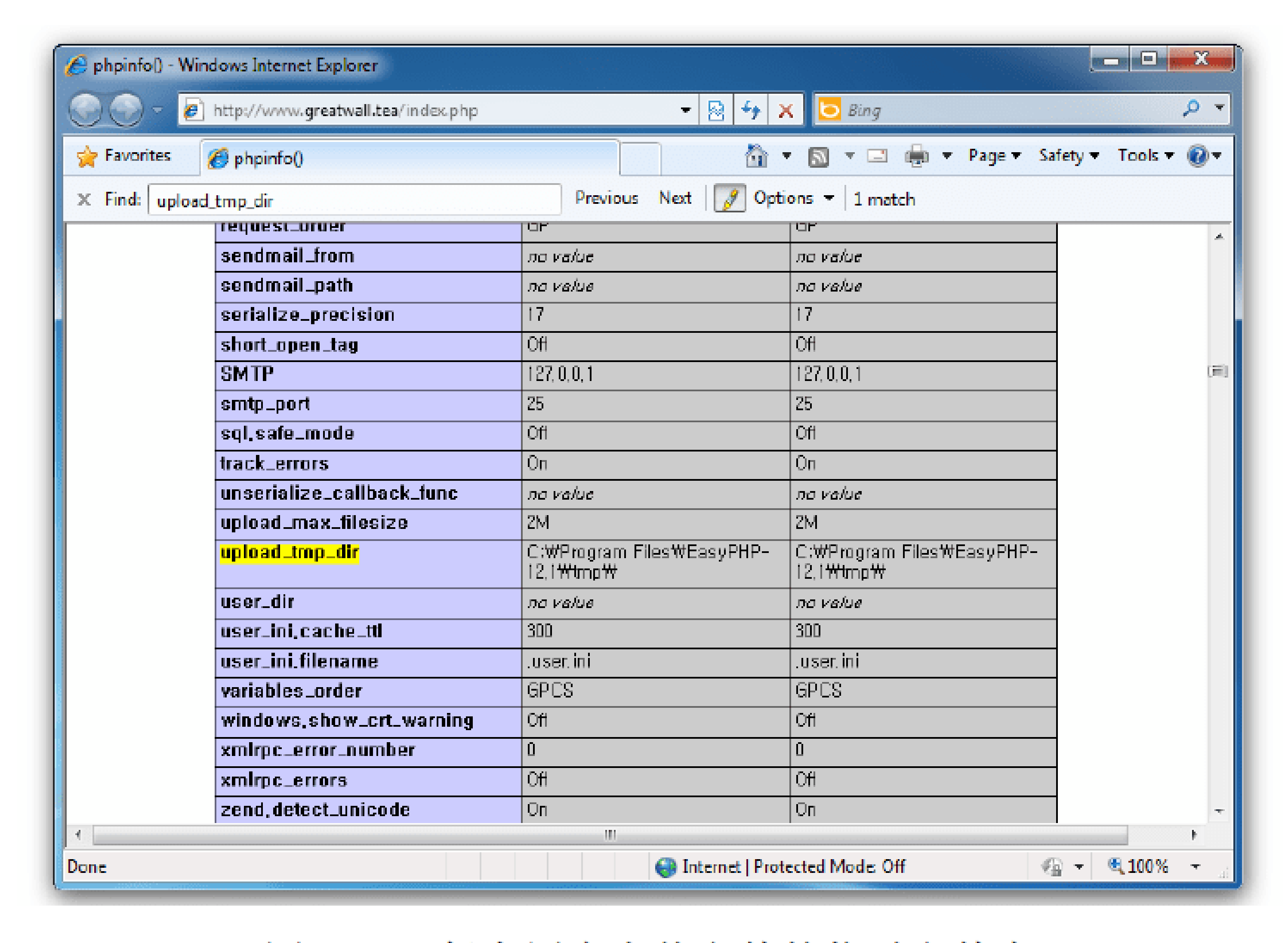


图 9-12　保存用户上传文件的临时文件夹

按照图 9-12 所示，使用的服务器保存用户上传文件的临时文件夹为“C:\Program Files\EasyPHP-12\tmp”。如果需要修改此文件夹的路径，可以打开 php.ini 文件，搜索“upload_tmp_dir”参数修改。

通常情况下，临时文件存放在哪里并不重要，一般不需要修改 php.ini 文件中“upload_tmp_dir”的值。

9.4.2　获取已上传文件的信息

在用户上传某个文件后，文件会被存放在一个临时文件夹内。这时，我们可以使用 PHP 脚本的$_FILES 数组来处理这个文件。注意$_FILES 是个二维数组：第一个维度是表单中文件域的名字，而第二个维度则是文件的相关属性。以例 9.6 中的表单所示，我们可以通过以下的方式获取已上传文件的相关属性：

```
echo $_FILES['user_file']['name'];
echo $_FILES['user_file']['type'];
echo $_FILES['user_file']['tmp_name'];
echo $_FILES['user_file']['size'];
```

在获取文件的相关属性后，就可以判断用户上传的文件是否符合要求。如果用户上传的文件符合我们的要求，就可以使用 move_uploaded_file()函数把文件转移到永久存放文件的文件夹内。

还是以例 9.6 中所示的表单为例，若用户上传的文件名为 testfile.txt，那么我们就可以使用如下的脚本将其移动到永久存放文件的文件夹（C:\data\）内，并将其重命名为 new_file.txt：

```
move_uploaded_file($_FILES['user_file']['tmp_name'],'C:\data\new_file.txt');
```

需要注意的是，这个永久存放文件的文件夹在存放文件之前必须已经存在，否则，上传文件失败。

下面，我们来看一个完整的例子。在这个例子中，需要定义一个表单用于上传图片。在用户单击“上传图片”按钮后。允许上传文件最大不能大于 500 KB。系统会对用户上传的文件进行检测，若文件大小超出大小限制或文件类型不正确，均显示上传失败。若文件大小和文件类型都符合要求，则上传成功。

【例 9.7】 文件上传：一个完整的例子。

```
<?php
   if(!isset($_POST['Upload'])){
       include("Ex9-06a.php");
    } else {
       if ($_FILES['image']['tmp_name'] == "") {
           echo "<b>Failed to upload the image. Check the file size.
               File must be less than 800 KB.</b>";
           include("Ex9-06a.php");
           exit();
       }
       if (!preg_match("/image/", $_FILES['image']['type'])) {
           echo "<b>The type of the current file is ".$_FILES['image']
           ['type']." and is not an image.
               Please select another one.</b>";
           include("Ex9-06a.php");
```

```
                exit();
            }
            else {
                $destination = dirname(__FILE__)." \image\\".$_FILES
                ['image']['name'];
                $temp_file = $_FILES['image']['tmp_name'];
                move_uploaded_file($temp_file, $destination);
                echo "<p><b>Succeeded in uploading the image:</b></p>
                    <p>{$_FILES['image']['name']} 22   ({$_FILES['image']
                    ['size']})</p>";
            }
        }
    ?>
```

在这段脚本中，我们需要注意的内容如下所示。

（1）使用 isset()函数来判断$_POST['Upload']是否已经赋值，这里的“Upload”为表单按钮的名称。如果用户单击过该按钮，那么$_POST['Upload']的值应该为定义该按钮时指定的值。

如果$_POST['Upload']的值还没有指定，也就是用户从来就没有单击过该按钮，那么系统就会使用 include()函数加载表单，提示用户选择需要上传的文件。如果用户已经单击过该按钮，那么$_POST['Upload']的值就已经设置，系统就开始判断用户指定的文件是否符合要求。

（2）另外，脚本里使用了两个 if 条件来判断用户上传文件的类型和大小。若其中一项不符合要求，系统就会执行 exit()函数退出脚本的执行。若两面都符合要求，则开始上传文件。

在上传文件之前定义了两个变量：$destination 和$temp_file，前者为文件上传后的永久存储路径，后者为文件上传后的临时存储路径。

（3）在定义文件上传后的永久存储路径时，我们使用了 dirname(__FILE__)函数来获取当前文件所处文件夹的路径。然后再加上专门用于存放图片的文件夹的名称。

需要注意的是，路径中使用的分隔符为“\”，与 PHP 中的转义字符相同。当我们在字符串中使用“\”时，该符号后的内容会被转回本义。因此，我们在专门用于存放图片的文件夹的名称后面加上了两个“\”，前一个为转义字符，后一个为路径分隔符。

现在来看看例 9.7 中引用的表单文件：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<meta content="text/html; charset=UTF-8" http-equiv="Content-Type">
<html>
    <head>
        <title>Upload Image</title>
    </head>
    <body>
        <ol>
            <li>Select the image you want to share with us.</li>
            <li>Click "Upload Image" button below after the
                path of the file appears in the text box.</li>
        </ol>
        <div align="center">
            <hr>
            <form enctype="multipart/form-data" action="Ex9-06b.php"
```

```
                method="POST">
                    <input type="hidden" name="MAX FILE SIZE" value="800000">
                    <input type="file" name="image" size="60">
                    <p></p>
                    <input type="submit" name="Upload" value="Upload Image">
                </form>
            </div>
        </body>
    </html>
```

在这份文件中，我们定义的表单中文件域的名称为“image”，提交按钮的名称为“Upload”，其值为“Upload Image”。

9.5　实战练习：记账工具（中）

在上一章的实战练习中，我们创建了一个 bookKeeping 类，用于逐条或批量添加收入和消费记录。在本章里，我们将规划记账工具的用户界面和数据传递方式。

9.5.1　界面预览

首先，来看看登录界面。在登录界面里，我们提供了两个文本框：一个让用户输入用户名，另一个让用户输入密码。然后，还提供了一个按钮，可以让用户提交登录。

在用户登录系统之后会直接进入添加支出记录的页面。在该页面的左侧是导航栏，用于引导用户分别进行“添加支出记录”、“添加收入记录”、“批量上传记录”和“查看记录列表”；而在页面的右侧则是功能区，用户可以通过填写表单完成相应的任务，如图 9-13～图 9-17 所示。

图 9-13　登录页面



图 9-14　添加消费记录



图 9-15　添加收入记录



图 9-16　批量导入记录

图 9-17　查看记录列表

9.5.2　脚本分析

这几个页面的 HTML 部分，我们就不在书中展示了，大家可以去看脚本。不过在这里，还是需要分析一下登录这个过程的。所谓登录，就是指用户在输入了正确的用户名和密码之后，可以获得授权，访问相应的页面。在这里，我们需要让用户在成功登录后，可以访问如图 9-13 到图 9-17 所示的页面。那么在这里，就要用上 Session 功能。

在登录页面上，PHP 脚本如下：

```
<?php
    session_start();
    if(isset($_REQUEST['login'])){
        $user = $_REQUEST['usr'];
        $pws = $_REQUEST['psw'];

        if(strlen($user) > 0) {
            $_SESSION['user'] = $user;
            header('location:./08.2_Book_Keeping_AddExpense.php');
        }
    }
?>
```

在上面这段脚本中，首先使用“session_start()”启用了 Session 功能。然后再判断用户是否提交了表单。若是，则获取用户提交的用户名和密码，并对其进行判断。在这里，为了简化代码，我们只判断用户是否输入了用户名，并据此决定是否将用户转向指定的页面。

脚本中使用了 header()函数，用来将用户转向指定的页面。需要注意的是，在使用 header()函数之前，不得向浏览器输出任何数据，否则 header 的转向功能不会起作用。

接下来再看看其他四个页面中的与 Session 相关的 PHP 脚本吧：

```
<?php
    session_start();
    include('07_Book_Keeping_Class.php');
    if(isset($_SESSION['user'])){
?>
```

```
/ * HTML 代码已略去 * /
<?php
    } else {
        header('location:./08.1_Book_Keeping_Login.php');
    }
?>
```

上面这段脚本同样也使用“session_start()”函数启用了 Session 功能。然后判断名为“user”的 Session 变量。若存在，则向浏览器输出上面略去的 HTML 代码；反之，则将用户转到登录页面。

最后，在批量导入记录时，需要上传模板文件。在下一章里将学习如何读取文件中的内容，因此，制订模板文件我们放在下一章的实战练习中讲解。在这一章里，只需要将用户填写好的模板文件上传到服务器上即可。

在 9.4 节里，我们学习了如何使用表单上传文件到服务器，现在我们在“批量导入记录”的页面上添加相应的脚本来获取上传后的文件：

```
<?php
    // 判断用户是否已提交表单
    if(isset($_REQUEST['batchUpload'])){
        // 获取用户上传的文件信息
        $temp_name = $_FILES['upload']['tmp_name'];
        $name = $_FILES['upload']['name'];
        $typeof = $_FILES['upload']['type'];
        // 判断用户上传的文件是否符合要求
        if($temp_name == '' ||
          !preg_match('/text/',$typeof)){
            $msg = '上传失败或格式不正确。';
        } else {
            // 移动文件到指定目录
            $destination = 'upload';
            move_uploaded_file($tmp_name, '$destination/$name');
            // 使用自定义类的 readTemp()方法来处理模板文件（见第 10 章）
            $msg = readTemp($name);
        }
    }
?>
```

至此，完成了为记账工具搭建可用界面的任务。在第 10 章里，我们将为页面添加相应的功能。

9.6　习　　题

（1）请编写一个 HTML 页面，用于收集用户的身份信息。主要包括：姓名、性别、出生年月以及照片。

（2）请编写一个 PHP 页面，用于处理用户通过表单提交的信息。在该页面上引用名为 class.authenticate.php 的类文件。

（3）请编写名为 class.authenticate.php 的类文件，用于验证用户提交的信息。

第 10 章　数据的存储

在第 9 章里，我们学习了如何使用表单从用户那里收集数据、如何使用 URL 传递数据到数据处理页面、如何使用 Cookie 缓存数据以及如何使用 Session 保证数据传输过程的安全。但似乎还缺少了很重要的一步，那就是如何将这些传递来的数据储存起来供以后使用呢？

数据库的功能很强大，相信很多同学都听说过数据库。只不过这个名字听上去就很有距离感，以至于很多听说过它的人也不见得会使用它。其实，PHP 只是个脚本语言，它与我们用什么方式存储数据没有关系。我们可以使用 PHP 脚本语言把数据存储到文本文件中，也可以把数据以结构化的方式存储到 XML 文件中，当然也可以把在需要存储的数据之间建立关联系以方便我们把数据存储到关系型数据库中。

在学习如何将数据存储到这些介质中之前。先来看看它们各自都有些什么优缺点：

（1）对于文本文件来说，小巧轻便应该是决定我们选择文本文件的关键词。和数据库相比，一个文本文件在存储同样多数据时，占用的磁盘空间更小，更容易被读取。但是需要注意的是它不能存储过多的数据，否则服务器在文件中查找起来的速度会异常的缓慢，进而会影响到整个站点的服务水平。打个容易理解的比方，就像我们在一间汗牛充栋的书房里找一篇文章一样。在没有任何的指引情况下，就只能对其展开地毯式搜索，人力、物力耗费都很大。

（2）对于 XML 文件来说，结构化应该是决定我们选择它的关键词。和文本文件相比，一个 XML 文件有着结构化的表现形式，对于存储在其中数据分门别类，有条有理。虽然 XML 存储同样多的数据时，会比文本文件占用的磁盘空间更大，但是它的数据结构决定了在其中查找数据的速度。就好像我们把刚才那间书房里的书分门别类的整理好了，想要找一本编程方面的书，那就只管按图索骥查询即可，肯定会比在杂乱无章的书房里找要快的多。不过，XML 文件存储的数据依然不能过量，一旦过量，系统反应也会慢下来，一样会影响到整个站点的服务水平。

（3）对于关系型数据库来说，数据关系应该是决定我们选择它的关键词。如果我们需要存储和反复使用的数据之间有着十分明显的逻辑关系，而且可预见的数据量十分巨大，那么最好使用关系型数据库，而不是上述两类存储介质。同时，数据库一般会有强大的安全能力，保证存储在其中的数据安全。这一点也是上述两种介质做不到的。

虽然文本文件和 XML 文件有些缺点，虽然数据库有着文本文件和 XML 文件无法比拟的优点，在选择存储介质时，最好量体裁衣，不要贪大求洋，以免造成浪费。

10.1　使用文本文件存取数据

我们在这里说的文本文件，顾名思义，指的是只包含文本的文件。如果读者使用的是

Windows 操作系统，那么选择“所有程序”|“附件”|“记事本”命令，然后创建一个文本文件。如果使用的是 Unix 操作系统，也可以使用操作系统中附带的类似于记事本的程序创建文本文件。

在我们的印象里文本文件应该指的就是 TXT 文件，这种看法比较狭隘。其实文本文件的扩展名无论是什么，它都应该可以用记事本或与之类似的程序读取。我们说 XML 文件也是一种文本文件，因为它只包含文本，而且可以用记事本程序来读取。在本节里，涉及到的文本文件，除了 TXT 文件之外，还有 CSV 和 TSV 文件，它们分别是 Comma Separated Values 和 Tab Separated Values 的缩写。通常用在不同的应用程序之间传递数据。

文本文件十分简单易用，不需要安装任何其他的程序，就可以用 PHP 脚本来读写文本文件。我们读写文本文件需要分三步走：第一步是打开文本文件、第二步是读取文本文件或往文本文件中写入数据、第三步则是关闭文本文件。这三个步骤虽然看上去和“怎么把一头大象放在冰箱里”的答案如出一辙，但它们缺一不可。

下面来看看如何用 PHP 脚本打开一个文本文件。

10.1.1　打开和关闭文本文件

按照之前说的，在读写文本文件之前，应该打开这个文本文件。我们可以使用如下的语句打开一个文本文件，并将这个文本文件存储到一个变量中。

```
$fh = fopen("filename", "mode")
```

在上面这条语句中使用了 fopen()函数。它带有两个必要参数，一个为文件名，另一个为模式。需要注意的是，在为 fopen()函数添加参数时，一定要用引号将这两个参数分别包裹起来。

另外，在打开一个文本文件时，一定要指定文件的打开方式。表 10-1 所示详细地罗列了 fopen()函数支持的文本文件打开方式。

表 10-1　fopen()函数支持的文本文件打开方式

模式	打开方式	说　　明
r	只读	如果指定的文件不存在，系统会显示一条警告信息
r+	读写	如果指定的文件不存在，系统会显示一条警告信息
w	只写	如果指定的文件不存在，PHP 会尝试创建该文件。如果指定的文件存在，PHP 会尝试覆盖该文件
w+	读写	如果指定的文件不存在，PHP 会尝试创建该文件。如果指定的文件存在，PHP 会尝试覆盖该文件
a	追加	如果指定的文件不存在，PHP 会尝试创建该文件。如果指定的文件存在，PHP 会尝试在此文件末尾追加数据
a+	读并追加	如果指定的文件不存在，PHP 会尝试创建该文件。如果指定的文件存在，PHP 会尝试在此文件末尾追加数据

我们在指定文件名时，可以只使用文件名，前提是该文件与执行读取该文件命令的脚本文件在同一个目录下。否则请指定该文件的路径，可以使用绝对路径或相对路径。如果需要读取的文件不在本地，也可以使用该文件的 URL。

现在以只读的方式打开一个文本文件：

```
$fh = fopen("file.txt", "r");
```

如果系统找不到文件 file.txt，或者这个文件根本就不存在，则系统会输出一条警告信息，就像下面这个信息一样：

```
Warning:fopen(file.txt):failed to open stream:No such file or directory in
C:\greatwall\chapter10\Ex10-01.php on line 2
```

虽然系统给出了警告信息，但那也仅仅是一条警告而已，不代表脚本停止运行了。脚本还会继续运行，只是如果后续的脚本中对有该文件的读写操作，它们都不会被执行。

为了避免这种情况，可以在打开该文本文件之前，使用 file_exists()函数判断一下该文件是否存在。因此，上面这段脚本可以改成：

```
1   <?php
2       if (file_exists("fileReadOnly.txt")) {
3           $fh = fopen("fileReadOnly.txt", "r");
4       } else {
5           die("The file does not exit.");
6       }
7   ?>
```

在脚本的第 2 行使用了 file_exists()函数，它只有一个必要参数，那就是文件名。如果这个文件在当前目录中存在的话，那么就会以文本流的形式存储在变量$fh 中；否则，系统会输出一条我们定义的信息。

现在，我们再以只写的方式打开一个文件。需要注意的是，当我们以只写的方式打开一个文本文件时，如果文件存在，该文件被覆盖；如果文件不存在，PHP 会创建该文件。

```
$fh = fopen("fileWriteOnly.txt", "w");
```

运行这段脚本后会发现，当前目录下出现了一个新的文本文件。该文本文件的名字正好是我们在脚本中定义的 fileWriteOnly.txt。

假若当前目录下有一个子目录为 subfolder。我们可以使用如下的脚本以只写的方式打开该子目录下的文本文件：

```
$fh = fopen("subfolder/fileWriteOnly.txt", "w");
```

当然，如果当前目录下并没有一个名为 subfolder 的子目录，则会出现如下的报错信息：

```
Warning: fopen(subfolder/fileWriteOnly.txt): failed to open stream: No such
file or directory in C:\greatwall\chapter10\Ex10-01.php on line 8
```

因此，在读取子目录中的文本文件之前，还是需要判断一下该子目录是否存在。这里，我们可以用到 is_dir()函数。上面这段脚本，就可以改成：

```
<?php
    if (is_dir("subfolder")) {
        $fh = fopen("subfolder/fileWriteOnly.txt", "w");
    } else {
        die("The directory does not exist.");
    }
?>
```

现在，我们再来打开一个远程的文本文件。所谓远程的文本文件，也就是说该文本文件不在本地。因此，建议读者以只读方式打开，并对该文件做读写操作。除非你确信你对

该文件有读写权限。

```
$fh=fopen("http://www.gutenberg.org/cache/epub/42304/pg42304.txt", "r");
```

上面这段脚本打开了位于古腾堡计划官方网站上的一本 TXT 格式的免费电子书《日本之密》（*The Gist of Japan*）。

如果运行这段脚本，没有系统报错的话，那么恭喜你成功打开了这本电子书。

在学会如何打开文本文件之后，现在要来看看如何关闭打开的文本文件。当使用 fopen()函数打开某个文本文件之后，该文本文件就以文件流的形式存放在某个变量中。如果需要关闭这个文件流，我们可以使用 fclose()函数。就像下面这段脚本一样：

```
fclose($fh);
```

在上面这段脚本中，使用的 fclose()函数只有一个必要参数，那就是存储了已经打开的文件流的变量。

现在总结一下本节中使用到的脚本。

【例 10.1】 打开和关闭文本文件。

```
<?php
    if (file_exists("fileReadOnly.txt")) {
        $fh = fopen("fileReadOnly.txt", "r");
    } else {
        die ("The file does not exit.");
    }

    /*只有当 fileReadOnly.txt 被成功打开，
    下面的脚本才会被运行。*/

    if (is_dir("subfolder")) {
        $fh = fopen("subfolder/fileWriteOnly.txt", "w");
    } else {
        die("The directory does not exist.");
    }

    /*只有当 subfolder 文件夹存在时，
    下面的脚本才会被运行。*/

    $fh= fopen("http://www.gutenberg.org/cache/epub/42304/pg42304.txt", "r");

    fclose($fh);
?>
```

10.1.2　向文本文件中写入数据

在以只写、读写、追加或读并追加的方式打开了一个文本文件之后，我们就可以向文本文件中写入数据了。在向文本文件中写数据时，会用到 fwrite()函数。

假如，我们需要向某个文本文件中追加今天的日期。我们可以这么写，如下所示。

【例 10.2】 向文本文件中写入数据。

```
<?php
    $today = date("Y-m-d");
    $fh = fopen("fileAppendTo.txt", "a");
```

```
    fwrite($fh, $today."\n");
    fclose($fh);
?>
```

上面这段脚本首先使用“追加”的方式打开了文件 fileAppendTo.txt。按照表 10-1 的说明，以“追加”方式打开一个文本文件时，若文件不存在，PHP 会尝试创建该文件。然后我们使用了 fwrite()函数向打开的文件中追加今天的日期。这个 fwrite()函数一共有两个必要参数，一个是存储着以文件流的形式打开的文本文件的变量，另一个是需要追加的内容。在追加完成后，用 fclose()函数关闭了该文件流。

若在运行这段脚本之前，fileAppendTo.txt 不存在。运行这段脚本之后，会发现当前目录下多了一个名为 fileAppendTo.txt 的文件。打开这个文件，里面赫然记载着今天的日期。如果反复运行这个文件，会发现越来越多的日期被追加到了文件末尾，就像下面这样：

```
2013-03-12
2013-03-12
```

10.1.3　从文本文件中读取数据

在向文本文件中写入了数据之后，我们的诉求就变得很明显了，那就是把这些数据读出来。从文本文件中读取数据，可以用到 fgets()函数。需要注意的是，这里的 gets 是第三人称单数形式，不是原型；另外，这个函数只有一个必要参数，那就是存储着以文件流的形式打开的文本文件的变量。这个 fgets()函数的返回值可能是整个文本文件，也可能是这个文本文件的某一行。到底是整个文本文件，还是这个文本文件的某一行，是由文本文件中的换行符，也就是在例 10.2 的 fwrite()函数中使用的“\n”决定的；如果脚本在读取数据时碰到了换行符，则 fgets()返回就是该换行符之前的内容；如果脚本一口气读到了文件末尾，那么 fgets()函数返回的就是整个文本文件的内容了。

比如，可以用下面这段脚本来读取例 10.2 中创建的 fileAppendTo.txt 文件。

【例 10.3】 从文本文件中读取数据。

```
<?php
    if (file_exists("fileAppendTo.txt")) {
        $fh = fopen("fileAppendTo.txt", "r");
        while (!feof($fh)) {
            $line = fgets($fh);
            echo $line;
        }
        fclose($fh)
    }
?>
```

上面这段脚本先是用 file_exists()函数判断 fileAppendTo.txt 文件是否存在。若该文件存在于当前目录下，我们就用 fopen()函数以只读的方式打开这个文件，并将该文件以文件流的形式存放于变量$fh 中。然后使用 while 语句循环输出该文件每一行的内容。

在 while 语句中，我们使用到了 feof()函数。这个函数是用来判断当前指针是否已经到达文件末端的。如果没有，就将当前行的内容输出给变量$line，然后输出$line 的内容。

上面这段脚本的运行的结果如下：

```
2013-03-12  2013-03-12
```

有的同学可能会问：这明明是两行的内容，为什么会显示在一行里呢？这个问题在之前的章节中已经讨论过了，大家可以再回想一下，我就不多言了。

现在来看看如何才能一小段一小段地读取文本文件里的内容。假如我们需要 4 个字符为一行地显示文本文件里存储的内容，可以把例 10.3 修改成如下的样子。

【例 10.4】 分段读取文本内容。

```
<?php
    if(file_exists("fileAppendTo.txt")) {
        $fh = fopen("fileAppendTo.txt", "r");
        while (!feof($fh)) {
            $piece = fgets($fh, 5);
            echo "$piece<br>";
        }
        fclose($fh);
    }
?>
```

在上面这段脚本中，fgets()函数指定了两个参数。一个为存储着以文件流的形式打开的文本文件的变量，另一个为一次需要读取的字符数。需要注意的是，脚本在实际读取数据时返回的字符数会比指定的字符数少一位。因此，在指定一次需要读取的字符数时，请务必加 1。

上面这段脚本输出的结果如下：

```
2013
-03-
12
2013
-03-
12
```

我们也可以把文本文件里的内容一行一行的输出到一个数组里，就像下面这样。

【例 10.5】 将文本文件中的数据读取到数组中。

```
<?php
    if(file_exists("fileAppendTo.txt")){
        $fh = fopen("fileAppendTo.txt", "r");
        while(!feof($fh)){
            $line[] = fgets($fh);
        }
        fclose($fh);
    }

    foreach ($line as $key => $value) {
        echo "[".$key."] ".$value."<br>";
    }
?>
```

如果大家还记得我们在讲数组的时候，提到的数组元素的遍历，那么应该对 foreach 这个语句有些印象。在这里先把 fileAppendTo.txt 中的内容读到了$line 数组中存储起来，然后用 foreach 语句遍历$line 数组。

上面这段脚本的运行结果如下：

```
[0] 2013-03-12
[1] 2013-03-12
```

如果我们需要一次性读取某个文本文件里的所有内容到一个变量中，那就不能继续使用 fgets()函数了。这时，需要使用 file_get_contents()函数，就像下面这样：

```
$content = file_get_contents("fileAppendTo.txt", 1);
```

这时，$content 的值就是一个字符串，而不是一个数组了。如果这时我们使用 echo 语句打印变量$content 的值，结果就应该是下面这个样子：

```
2013-03-12  2013-03-12
```

10.1.4　从 CSV 和 TSV 文件中读取数据

文本文件存储的数据除了可以分行以外，似乎显得有些杂乱无章。而当我们从一个文本文件中把数据读取出来以后，需要把它转存到另外一个文件中时，如果这个文件对导入的数据有格式上的要求，那单纯的 fopen()、fgets()和 fclose()函数就有点力不从心了。事实上，在这一节的开头，我们曾经提到了 CSV 和 TSV 文件，知道它们通常会被用来在不同的应用程序间传递数据。根据这一点可以推断：PHP 应该会提取 CSV 文件或 TSV 文件中的数据的方法。

那么什么是 CSV 文件呢？

CSV 文件是一种有着简单格式的文本文件。顾名思义，这种类型的文本文件里存储着用逗号分隔开来的一系列的值，就像下面这样：

```
John Smith, 1234 Oak St., London, GB, 999990
Wang Xiaoer, A28 Shangdi Sanjie St., Beijing, PRC, 100085
```

打开记事本，将上面这两行内容复制到记事本中，然后选择“文件”|“另存为…”命令。在弹出的对话框中，文件格式选择“所有文件”，然后在文件名中输入“addressbook.csv”，单击“保存”按钮。一个 CSV 文件就做好了。

如果你的电脑上还安装了 Microsoft Office 或者金山的 WPS 这样的办公软件，那么恭喜你。你还可以用它们的电子表格软件打开这个后缀名为 CSV 的文件，如图 10-1 所示。

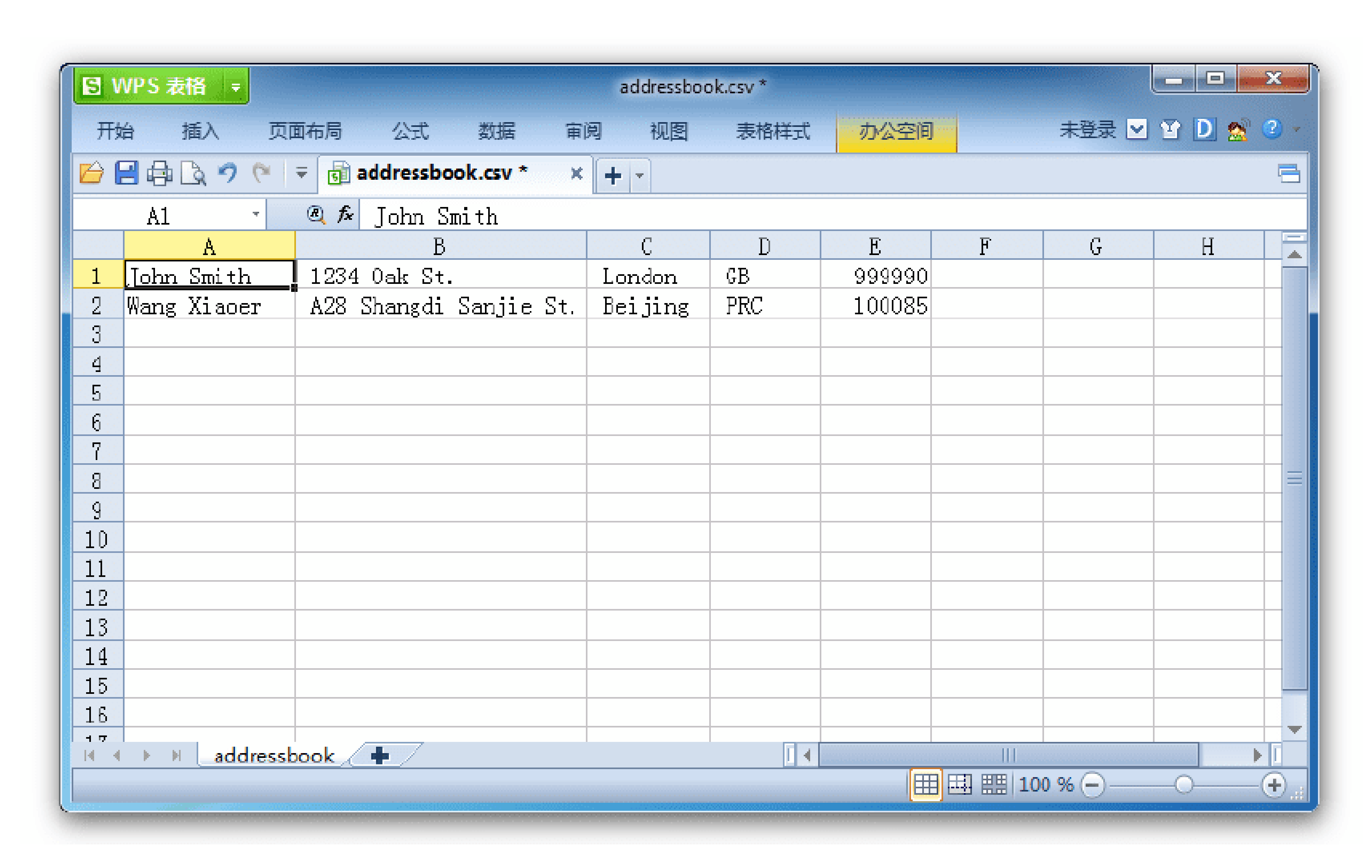


图 10-1　用电子表格软件打开后缀名为 CSV 的文本文件

我们也可以用例 10.6 中的脚本来创建一个 CSV 文件。

【例 10.6】 用 PHP 创建 CSV 文件。

```
<?php
    /* 首先我们以追加模式打开一个文件，以防
       该文件在当前目录下不存在 */
    $fh = fopen("fileCSV.csv", "a");

    /* 然后我们可以创建一个数组，用来存放那些
       需要存放到该 CSV 文件中的数据 */
    $address[] = "John Smith, 1234 Oak St., London, GB, 999990";
    $address[] = "Wang Xiaoer, A28 Shangdi Sanjie St., Beijing, PRC, 100085";
    $address[] = "Liu Laipin, 1345 Yiyuan St., Wuhan, PRC, 430001";

    /* 第三步，我们需要使用 for 循环来将数组中存放的数据
       写入到 CSV 文件中*/
    for($i = 0; $i < 3; $i++){
        fwrite($fh, $address[$i]."\n");
    }

    // 最后，关闭打开的文件
    fclose($fh);
?>
```

运行了上面这段脚本后，当前目录下生成了一个名为 fileCSV.csv 的文件。打开该文件，你会发现$address 数组中的数据被安安稳稳地存放到了文件中。

现在，我们就来从这个 CSV 文件中读取数据。如果还是使用 fgets()函数的话，那么就只能一行一行地读了。因此 CSV 文件和普通的文本文件相比而言的优势就不复存在了。为了解决这个问题，PHP 提供了一个新的函数：fgetcsv()。这个函数有两个必要参数，一个为文件名，另一个为读取的最大字符数。

【例 10.7】 从 CSV 文件中读取数据。

```
<?php
    // 检查 fileCSV.csv 文件是否存在
    if(file_exists("fileCSV.csv")){
        $fh = fopen("fileCSV.csv", "r");
        $info = fgetcsv($fh, 1000);
        foreach ($info as $key => $value) {
            echo "\$info[".$key."] ".$value."<br>";
        }
        fclose($fh);
    }
?>
```

在例 10.7 的这段脚本中，首先判断了 fileCSV.csv 文件是否存在于当前文件夹下。如果存在，则以只读的方式打开该文件，然后用 fgetcsv()函数将文件的前 1000 个字符读取到$info 数组中。需要注意的是 fgetcsv()函数返回的是一个数组。然后用 foreach()函数遍历了数组$info 的值。

运行这段脚本后，得到的结果如下：

```
$info[0] John Smith
$info[1] 1234 Oak St.
$info[2] London
```

```
$info[3] GB
$info[4] 999990
```

原来，fgetcsv()函数只读取了第一行的内容，并且将用逗号分隔开的值存放到了$info数组的每个元素当中。如果我们想遍历该文件中的所有内容，就得将上面的这段脚本改成：

```
<?php
    //检查 fileCSV.csv 文件是否存在
    if(file exists("fileCSV.csv")){
        $fh = fopen("fileCSV.csv", "r");

        //Read the data saved in the CSV file.
        $fh = fopen("fileCSV.csv", "r");

        $i = 0;
        while (!feof($fh)) {
            $info = fgetcsv($fh, 1000);
            if($info <> FALSE) {
                foreach ($info as $key => $value) {
                echo "[$i][$key] $value<br>";
                }
                $i++;
            } else {
                die("Reached the end of the file.");
            }
        }
        fclose($fh);
    }
?>
```

上面这段脚本还有一点需要说明，那就是 while 循环中那个 if 语句的条件。为什么要有这么一句呢？因为文本文件中很有可能会存在空行，如果一旦碰到空行的话，那么我们使用 fgetcsv()函数得到的就不是个数组了，因而也就不能使用 foreach 来遍历了。如果强行使用 foreach 来遍历不是数组的元素，则得到了就只能是系统的警告。因此，在遍历数组之前，一定要保证遍历的对象是个数组，而“$info <> FALSE”就是用来保证这一点的。

最后来看看如何读取 TSV 格式的文本文件吧。

其实，fgetcsv()函数不光能读取 CSV 格式的文本文件，还可以读取所有*SV 格式的文件。因为这个函数的第三个参数就是分隔符，而 TSV 文件里的值都是以“\t”来分隔的。在键盘上它有一个对应的键，那就是 Tab 键。在记事本里，如果按一下 Tab 键，光标就会缩进一段距离。我们可以按照制作 CSV 文件的方式制作一个 TSV 文件，只需要把逗号换成“\t”即可。

之所以要使用 TSV 文件的根本原因是，存储在文本文件的数据里可能真的会用到逗号。如果一旦如此，而我们又将这些数据存放在了一个 CSV 文件里，那某一条数据可能就会被拆成两截了。

10.1.5　实战练习：用文本文件做数据源的留言本

【目标】：制作一个以文本文件为数据源的留言本。

【效果】：制作的效果如图 10-2 所示。

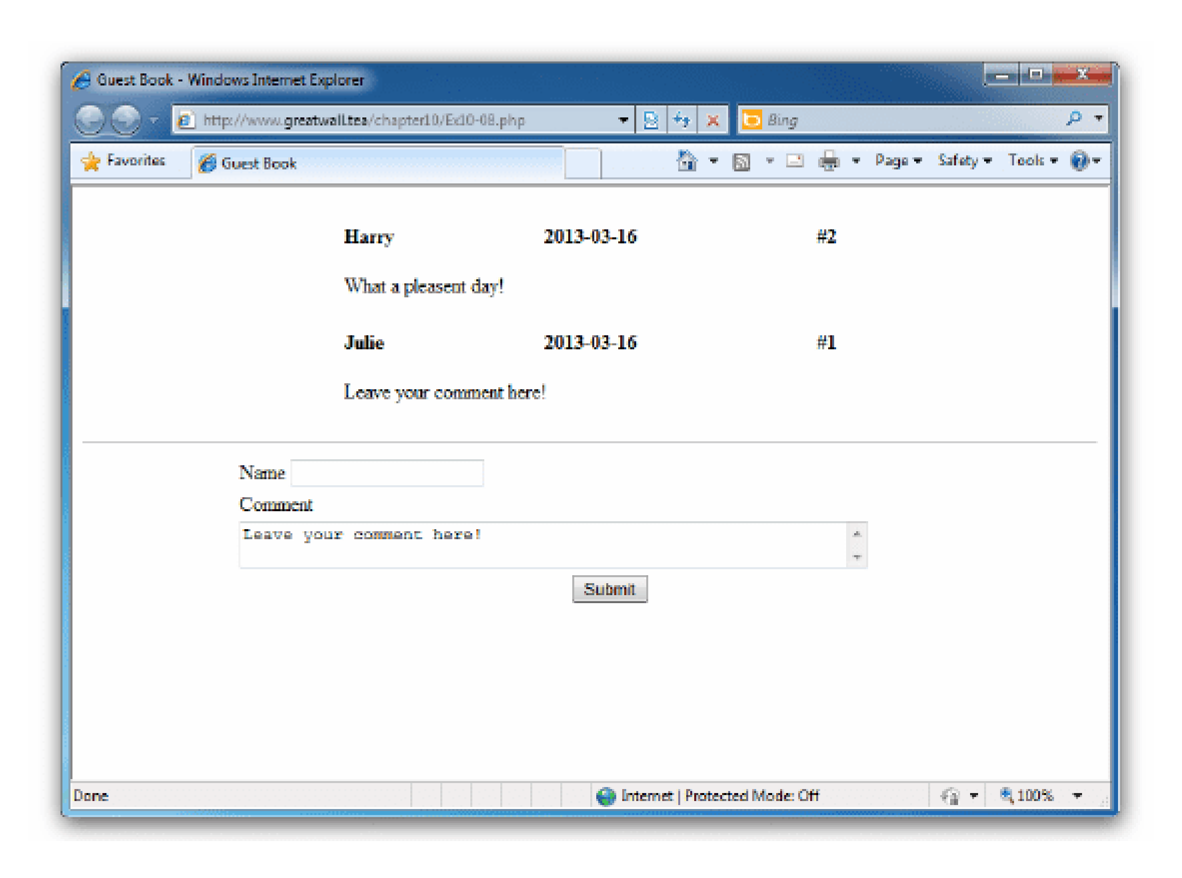


图 10-2　以文本文件为数据源的留言本

【要求】

- ❑ 使用所学的内容做一个以文本文件为数据源的留言本；
- ❑ 这个留言本可以记录留言人的名字、留言的时间和留言的内容；
- ❑ 留言需按照从新到旧的顺序显示。

【源代码】

```
<?php
    /*
     * 检查是否有用户提交了留言
     * 若是，以追加的方式打开一个名为 guestBook.tsv 的文件
     * 并在其末尾追加用户提交的留言。
     */
    if (isset($_REQUEST['append'])) {       //判断用户是否提交了留言
        $fh_append = fopen("guestBook.tsv", "a");
        $comment = $_POST['userName']."\t".date("Y-m-d")."\t".$_POST
        ['Comment']."\n";
        fwrite($fh_append, $comment);
        fclose($fh_append);
    }

    /*
     * 1.   若名为 guestBook.tsv 的文件存在，打开该文件，
     *   否则创建一个变量$message 用来存储告知用户
     *   该文件不存在的消息。
     * 2. 用 fgetcsv() 函数分行读取 TSV 文件中的内容，并
     *   把它们存入一个名为$comment 的数组。
     * 3. 创建三个数组，分别将其命名为 $commentUser,
     *   $commentDate, 和 $commentCon 分别对应一个用
     *   户的名字、用户留言的日期和用户的留言。
     * 4. 变量$i 用来存储 TSV 文件中数据的行数。每读一行
     *   变量$i 的值加 1。
     */
    $i = 0;
    if (@$fh_read = fopen("guestBook.tsv", "r")) {
        while (!feof($fh_read)) {
            $comment = fgetcsv($fh_read, 1000, "\t");
            if ($comment <> FALSE) {
                $commentUser[$i] = $comment[0];
                                                //三个数组分别存储用户的姓名
```

```
                    $commentDate[$i] = $comment[1];//用户留言的日期
                    $commentCont[$i] = $comment[2];//用户留言的内容
                    $i++;                    //每读取一行的记录，变量$i 加 1
                }
            }
        } else {
            $message = "The guestbook does not exist.";
        }
    ?>
    <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
    <meta content="text/html; charset=UTF-8" http-equiv="Content-Type">
    <html>
        <head>
            <title>Guest Book</title>
            <style>
                #myForm {
                    width: 50%;
                    margin: 0 15%;
                }

            </style>
        </head>
        <body>

            <br>
            <center>
                <?php
                /*
                 * 1. 若变量$i 大于 0，则 TSV 文件中有数据，反之
                 *   则打印变量$message，告知用户该文件不存在
                 * 2. 用 for 循环来遍历上面创建的三个数组，并按照
                 *   从新到旧的顺序罗列它们
                 */
                    if ($i > 0) {   //若变量$i 为零，输出变量$message 的值
                        for ($j=$i-1; $j>=0; $j--) {
                                    //以从新到旧的方式显示留言记录
                ?>
                <table border="0" width="400">
                    <tr>
                        <td align="left" width="30%">
                                //输出当前指针所指留言记录对应的用户名
                            <strong><?php echo $commentUser[$j]; ?>
                            </strong>
                        </td>
                        <td align="center" width="40%">
                                //输出当前留言记录对应的留言日期
                            <strong><?php echo $commentDate[$j]; ?>
                            </strong>
                        </td>
                        <td align="right" width="30%">
                                //输出当前留言记录对应的楼层数
                            <strong><?php echo "#".($j+1); ?></strong>
                        </td>
                    </tr>
                    <tr>
                        <td colspan="3" align="left">
                            <p></p>      //输出当前留言记录对应的留言内容
                            <p><?php echo $commentCont[$j]; ?></p>
```

```
                    </td>
                </tr>
                <tr><td colspan="3" align="center"></td></tr>
            </table>
            <?php
                    }
                } else {
                    echo $message;
                }
            ?>
        </center>
        <hr />
        <div id="myForm">
            <?php
                /*
                 * 关于这个表单，我们在 action 字段里指定的文件名后
                 * 加上了一个问号和一个字符串，用来判断用户是否提
                 * 交了留言
                 */
             ?>
            <form action="Ex10-08.php?append" method="post">
                <table border="0" width="600 px">
                    <tr>
                        <td width="60%" align="left">
                            <label>Name</label>
                            <input type="text" name="userName" />
                        </td>
                    </tr>
                    <tr>
                        <td align="left">
                            <label>Comment</label>
                        </td>
                    </tr>
                    <tr>
                        <td align="left">
                            <textarea name="Comment" cols="60">
                                Leave your comment here!</textarea>
                        </td>
                    </tr>
                    <tr>
                        <td align="center">
                            <input type="submit" name="Submit"
                            value="Submit" />
                        </td>
                    </tr>
                </table>
            </form>
        </div>
    </body>
</html>
```

10.2　使用 XML 存取数据

上一节学习了在文本文件中存储数据和从文本文件中读取数据。在本节里，我们来看看如何在 XML 中存储数据和从 XML 文件中读取数据。如果对 XML 不太了解，或者不想

使用 XML 来存储和读取数据的话，可以跳过本节的内容，直接阅读 10.3 节，使用数据库存取数据。

在学习如何使用 XML 存取数据之前，先来看看什么是 XML。

XML，全名叫可扩展标记语言（Extensible Markup Language）。按照 W3C 的说法，XML 是一种脱生于 SGML 的非常简单、灵活的格式。最初，XML 被设计来解决大型电子出版业务当中遇到的挑战。而现在，XML 在种类繁多的应用程序间进行数据交换当中发挥的作用也日渐重要。

XML 有如下几个特点：

- 它是纯描述类的标记语言。不带任何样式标记，只是用来以结构化的方式存储和传输信息。
- 它是以纯文本的形式存储的。有能力处理纯文本的软件都可以处理 XML。
- XML 中的标签是用户自己定义的。标签之间的关系可以通过它们之间的层级结构得到展现。

如果读者对 XML 感兴趣，也可以自行搜索相关的内容与教程。从现在开始，我们假设读者已经了解了 XML 是什么以及如何编写 XML 文件。在此基础上，我们应该要使用 PHP 来操作 XML，即构造 XML 从 XML 文件中读取数据，向 XML 文件中写入数据等。

10.2.1　加载和读取 XML 数据

我们可以从 XML 格式的字符串中读取 XML，也可以从一个 XML 文件中读取 XML。通常的做法是将 XML 存放在一个 XML 文件中，然后用 simplexml_load_file()方法将其读取到 simpleXMLElement 对象中。

在例 10.9 中，先创建一个 XML 文件，将命名为 notes 并保存在当前目录下。其内容如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<notes>
    <note id="1">
        <from>Jimmy Carter</from>
        <to>Routney Dickinson</to>
        <date>2013-03-12</date>
        <subject>Hello, Routeney!</subject>
        <noteBody>
            Hi! Old buddy! It's about five years since we last met each other!
            I'll have a trip to Boston tomorrow, and would like to meet you there!
            Let me know if you are free then.
        </noteBody>
    </note>
</notes>
```

然后，可以使用下面的语句将这个文件的内容读取到一个变量中。

【例 10.9】 从 XML 文件中读取数据。

```
<?php

    /*
     * 在这里我们可以用 var_dump()或 print_r()函数
```

```
5        * 来检验$xmlObject 是否是一个 simpleXMLElement 对象
6        */
7       $xmlObject = simplexml_load_file("notes.xml");
8
9       foreach ($xmlObject->children() as $block) {
10
11          // 获取子节点的文本和属性
12          echo "<strong>".$block->from."</strong> send <strong>"
13          .$block->to."</strong> a note IDed <strong>".$block['id'].
14          "</strong> on <strong>".$block->date."</strong>.<br>";
15          echo "<hr />";
16
17          // 像遍历数组一样遍历一个节点
18          foreach ($block->children() as $tag => $text) {
19              if ($tag <> "noteBody") {
20                  echo "<strong>".$tag."</strong>: ".$text;
21                  echo "<br>";
22              } else {
23                  echo "<p>".$text;
24              }
25          }
26          echo "<br><br>";
27      }
28  ?>
```

上面这段脚本首先使用 simplexml_load_file()函数加载了 notes.xml 文件，并将其赋值给了变量$xmlObject。按照之前的说法，变量$xmlObject 是个 simpleXMLElement 对象。然后用 foreach 语句遍历这个对象。在遍历对象时，使用了 simpleXMLElement 类的 children()方法来获取对象$xmlObject（也就是 note.xml 的根节点）的子节点。这个 children()方法在遍历某节点下的所有子节点是非常有用的。

除此之外，还可以使用元素选择器“->”来获取存储在某个节点的下文本。在上面这段脚本的第 12～14 行，变量$block 对应的是该 XML 文件中的“**<note>**”节点。我们分别使用了“**$block->to**”、“**$block->from**”和“**$block->date**”获取了收件人、发件人和日期三个节点下存储的文本信息。同时，我们还使用“**$block['id']**”对应的“**<note>**”节点下的属性“id”的值。

这段脚本输出的内容如图 10-3 所示。

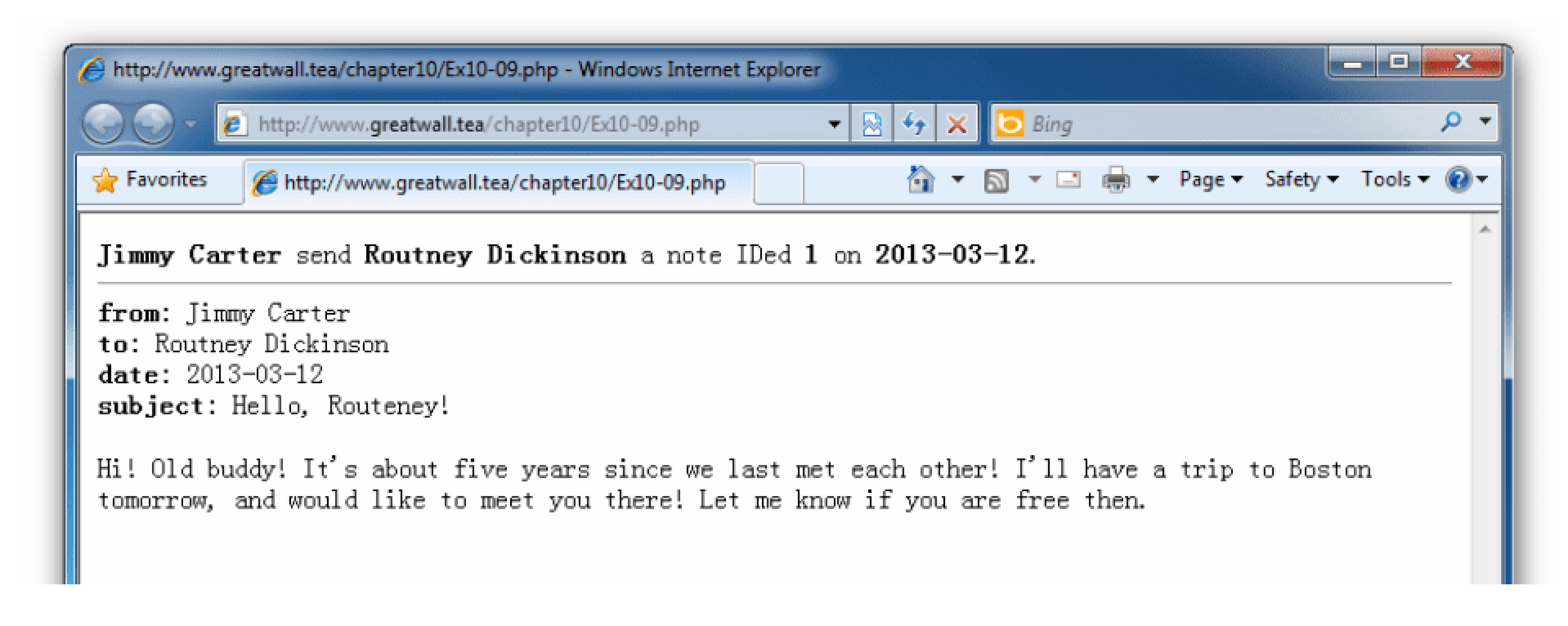


图 10-3　加载和读取 XML 数据

例 10.9 中使用的 notes.xml 文件其实也可以用字符串的形式存储到一个变量中，就像下面这样：

```
$xmlString = <<<XML
<?xml version="1.0" encoding="UTF-8" ?>
<notes>
    <note id="1">
        <from>Jimmy Carter</from>
        <to>Routney Dickinson</to>
        <date>2013-03-12</date>
        <subject>Hello, Routeney!</subject>
        <noteBody>
            Hi! Old buddy! It's about five years since we last met each other!
            I'll have a trip to Boston tomorrow, and would like to meet
            you there!
            Let me know if you are free then.
        </noteBody>
    </note>
</notes>
XML;
```

然后使用 simplexml_load_string()方法读取这个字符串，其他的所有操作方法都是一样的。需要说明的是，simplexml_load_string()和 simplexml_load_file()两个函数返回的都是 simpleXMLElement 对象。

10.2.2　修改 XML 文件中的数据

在学习了如何加载和读取 XML 文件中的数据之后，再来看看如何修改 XML 文件中的数据。

【例 10.10】 修改 XML 文件中的数据。

```
<?php
    //读取文件 notes.xml 中的 XML 数据
    $xmlObject = simplexml_load_file("notes.xml");

    //设置节点“to”的值
    $xmlObject->note->to = 'Anthony Tsu';

    //打印修改后 XML 数据
    $xmlString = $xmlObject->asXML();
    echo $xmlString;

    //以只写方式打开 notes.xml 文件
    //并用$xmlString 的值覆盖原有数据
    $fh = fopen("notes.xml", "w");
    fwrite($fh, $xmlString);
    fclose($fh);
?>
```

运行了上面这段脚本后，我们再运行例 10.9 中的那段脚本，就会发现 to 节点的值真的发生了如图 10-4 所示的变化。

在例 10.10 这段脚本里，我们使用 simplexml_load_file()读取到变量$xmlObject 中。这时，变量$xmlObject 是一个 simpleXMLElement 对象。紧接着，我们使用元素选择符“->”选择“to”节点，并将它的值修改成了“Anthony Tsu”。然后，使用 simpleXMLElement 对象的 asXML()（也可以写成 saveXML()）方法将这个修改后的 simpleXMLElement 对象转换成一个字符串并将这个字符串赋值给了变量$xmlString。最后，我们使用了 fopen()、

fwrite()和 fclose()方法将变量$xmlString 的值以覆盖的方式保存到了 notes.xml 中。

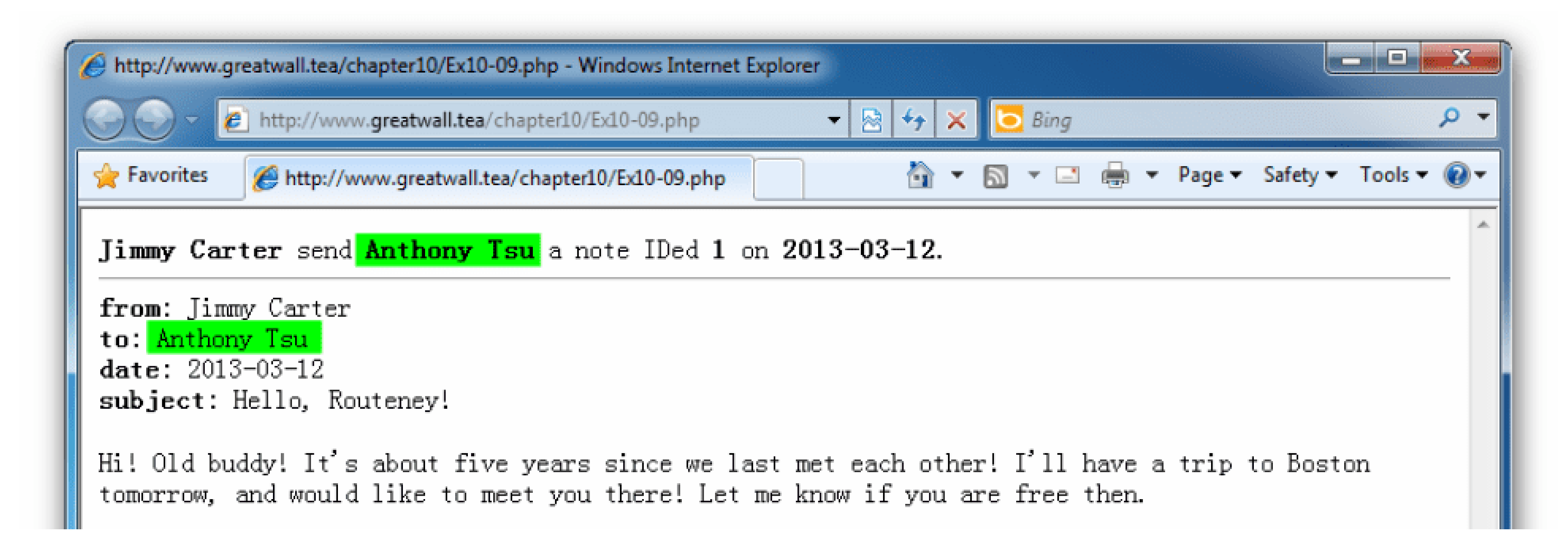


图 10-4　修改 XML 中的数据

于是，“to”节点的值由原来的“Routney Dickinson”变成了现在的“Anthony Tsu”。

10.2.3　向 XML 文件中添加数据

现在，我们再试着向 notes.xml 中添加一条记录，内容如表 10-2 所示：

表 10-2　文件notes.xml中的一条新记录

From	Samuel Johnson
To	Annie Hathaway
Date	2013-3-17
Subject	Reminder
Body	Don't forget the meeting at 2 p.m. this afternoon.

从表 10-2 所示中列出的数据来看，notes.xml 文件中的第二条记录是一个名叫 Samuel Johnson 的人向另一个名叫 Annie Hathaway 的人发出的提醒消息。消息的内容是让 Annie 不要忘记参加今天下午两点的会议。

为了将这批数据存储到 notes.xml 文件中，例 10.11 展示的脚本使用到 simpleXMLElement 对象的 addAttribute()、addChild 和 count()方法。其中，count()方法返回的是 XML 数据中当前节点拥有的一级子节点数；而 addAttribute()和 addChild()都拥有两个参数：第一个参数为节点标签，第二个参数为节点值。

【例 10.11】 向 XML 文件中添加数据。

```
<?php
    //读取文件 notes.xml 中的 XML 数据
    $xmlObject = simplexml_load_file("notes.xml");

    //计算当前存储的 note 记录的 ID
    $count = $xmlObject->count() + 1;

    //在根节点 notes 下添加一个 note 节点，并将其赋值给变量$note
    $note = $xmlObject->addChild("note");

    //为 note 节点添加属性“id”，并将变量$count 的值赋给它
    $note->addAttribute('id', $count);

```

```
    //为 note 节点添加 “to” 等五个子节点
    $note->addChild('from', 'Samuel Johnson');
    $note->addChild('to', 'Annie Hathaway');
    $note->addChild('date', '2012-03-17');
    $note->addChild('subject', 'Reminder');
    $note->addChild('noteBody', 'Don\'t forget the meeting at 2 p.m. this afternoon.');

    //将修改后的 XML 数据以字符串的形式赋值给变量$xmlString
    $xmlString = $xmlObject->asXML();

    //将变量$xmlString 的值以覆盖的方式存储到文件 notes.xml 中
    $fh = fopen("notes.xml", "w");
    fwrite($fh, $xmlString);
    fclose($fh);

    //引入例 10.9 中的脚本
    include('Ex10-09.php');
?>
```

注意，不要重复运行该脚本。否则这一条数据会以不同的 ID 重复写入 notes.xml 文件中。上面这段脚本运行的结果如图 10-5 所示。

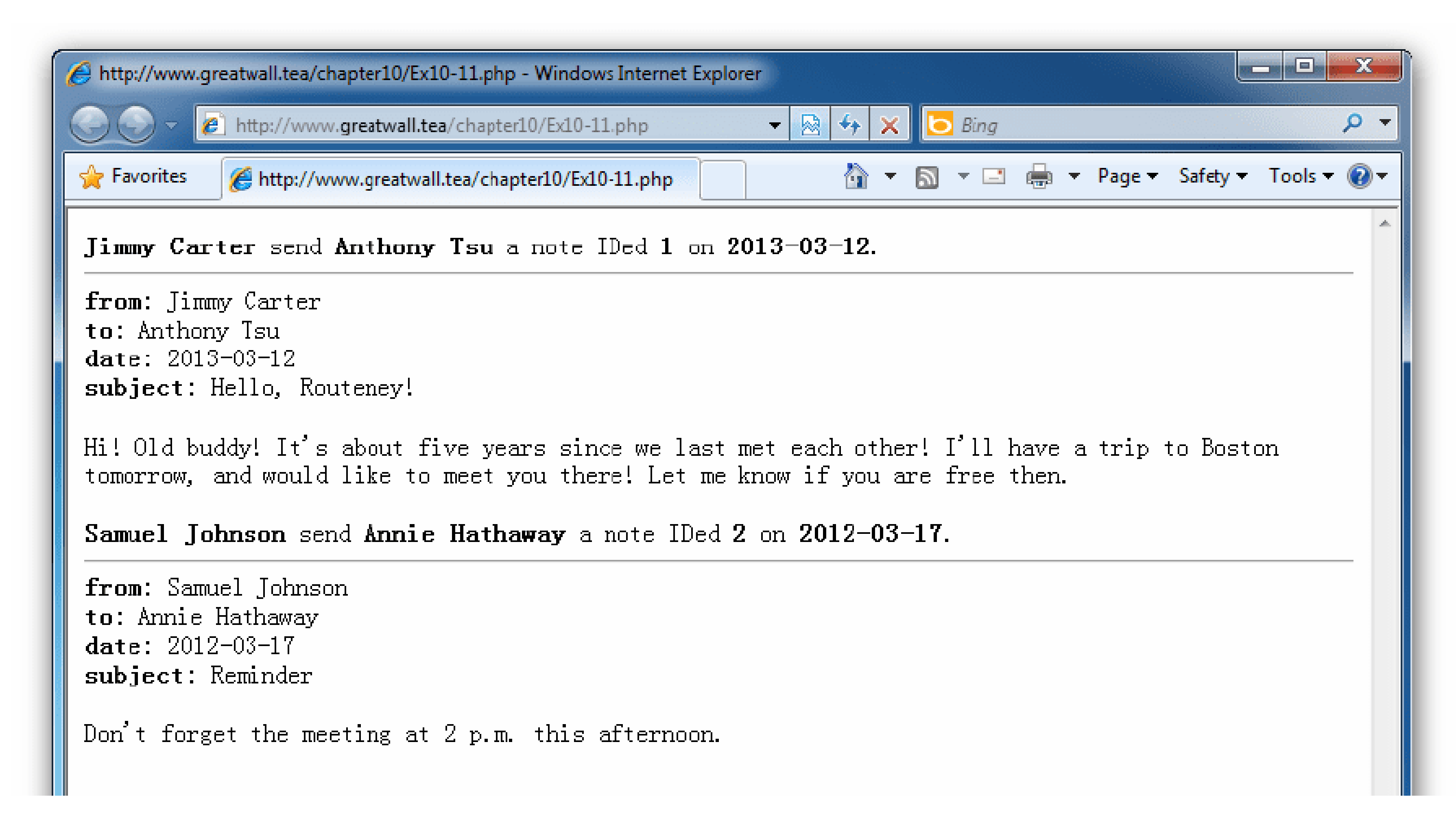


图 10-5　向 XML 文件中添加数据

例 10.11 的这段脚本，使用了 SimpleXMLElement 对象的 addAttribute 和 addChild 方法。我们知道$xmlObject 变量中存储的是从 notes.xml 文件中读取的所有 XML 数据。也就是说当前这个 SimpleXMLElement 对象的指针指向的是根节点，也就是“notes”节点。当在该节点下执行 addChild()方法时，就会在该节点下添加一个指定的子节点。简而言之，addChild()方法可以帮助我们在当前节点下创建一个指定的子节点。而 addAttribute()也是类似的，可以在当前节点下为其创建一个指定的属性。

10.2.4　遍历 XML 文件中的数据

之前提到的 simplexml_load_file()函数，其实它的第二个参数就是指定将文件中的内容读取到什么类型的对象中，这个参数的默认值是 SimpleXMLElement 对象。如果我们将其

指定为 SimpleXMLIterator，那么就可以使用 SimpleXMLIterator 对象提供的遍历数据的方法，轻松方便地在 XML 各元素间游走了。

来看一个例子。

【例 10.12】 遍历 XML 文件中的数据。

```
1   <?php
2       $xmlObject = simplexml_load_file('notes.xml', 'SimpleXMLIterator');
3       $xmlObject->rewind();
4
5       $i=0;
6       while($i<$xmlObject->count()){
7           foreach ($xmlObject->current() as $key => $value) {
8               echo $key.': '.$value.'<p>';
9           }
10          $xmlObject->next();
11          $i++;
12      }
13  ?>
```

在这段脚本中，我们使用 simplexml_load_file()函数的 SimpleXMLIterator 参数把 notes.xml 文件中的内容读取出来赋值给了变量$xmlObject。这时变量$xmlObject 就是一个 SimpleXMLIterator 对象。SimpleXMLIterator 类是由 SimpleXMLElement 扩展而来，继承了 SimpleXMLElement 类的所有属性和方法。这也是为什么我们可以在脚本的第 6 行使用 count()方法的原因。

在上面这段脚本中，除了 count()方法之外，其他的方法都是 SimpleIterator 特有的方法。其中，rewind()方法用于移动当前指针到第一个元素。注意，我们只有执行了 SimpleXMLIterator 对象的 rewind()方法后，执行其他的方法才会有结果，否则得到的结果一律为“NULL”。接下来，current()方法返回当前节点的内容，next()方法将指针移动到下一节点。除此之外，SimpleXMLElement 对象还有诸如 getChildren()这个用来获取当前结点下所有子节点的方法、hasChildren()这样判断当前节点下是否拥有子节点的方法，以及 valid()这样用来判断当前节点是否为一个合法节点的方法。

关于这些方法的具体内容，读者可以参考 PHP 手册中关于操作 XML 数据的相关内容。可以访问如下网址：

```
http://www.php.net/manual/en/book.simplexml.php。
```

SimpleXMLIterator 类通常是在对 xPath 语法不太了解的情况下，需要操作 XML 数据时使用的。如果你对 xPath 有所了解，也可以尝试用 xPath 来操作 XML 数据。这时，就没有必要使用 SimpleXMLIterator 对象了。因为 SimpleXMLElement 类为我们提供了使用 xpath 定位 XML 元素的方法。

来看一个例子。

【例 10.13】 SimpleXMLElement 类的 xpath()方法。

```
<?php
    $xmlObject = simplexml_load_file('notes.xml');

    // Obtains the children of the 'note' node
    $notes = $xmlObject->xpath('//note');

    //Note that the return value of the xpath() method is an array
```

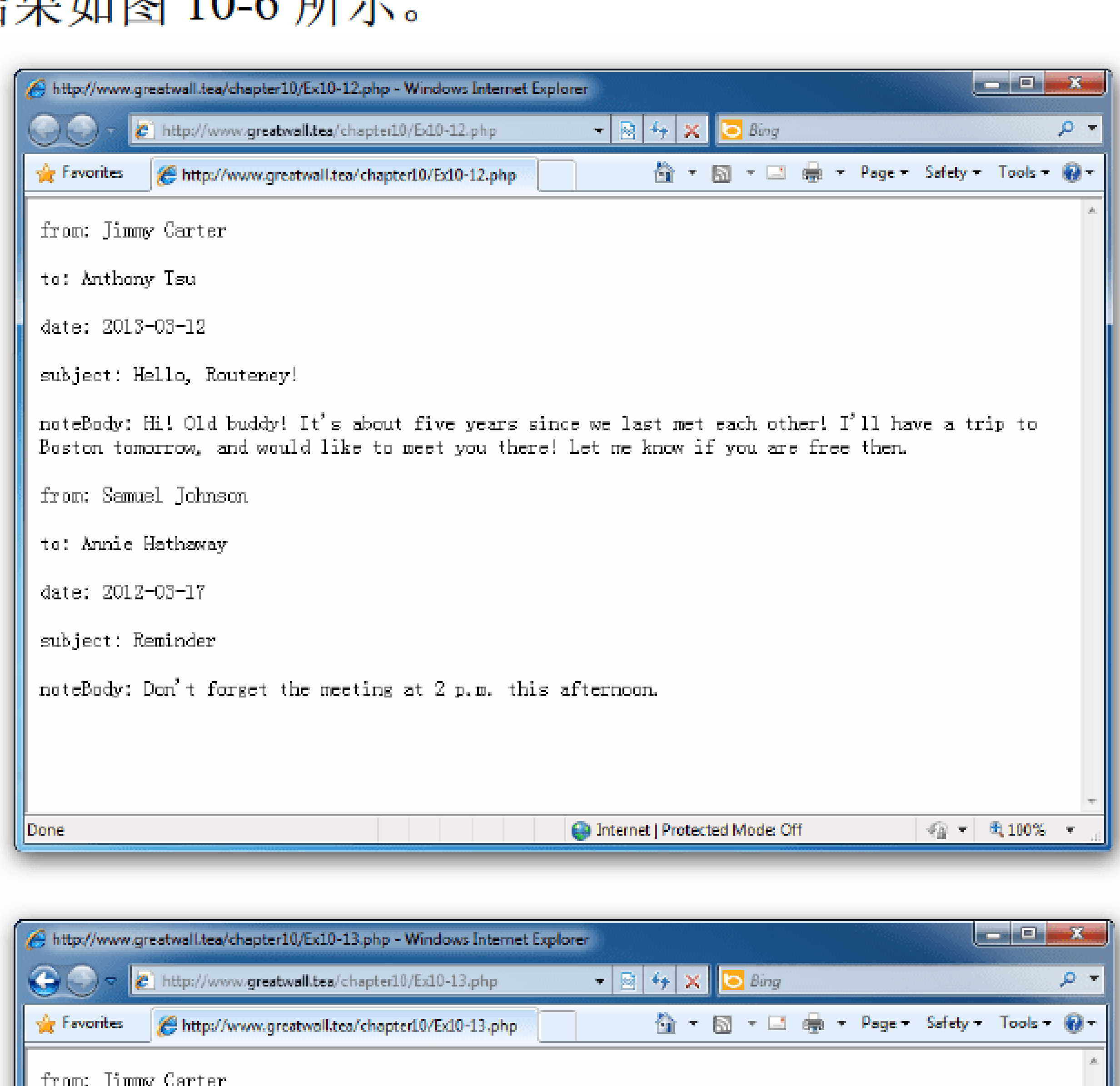
```
        foreach ($notes as $note) {
            foreach ($note as $key => $value) {
                echo $key.': '.$value.'<p>';
            }
        }
    ?>
```

在上面这段脚本中，我们使用了 simple_load_file()函数的默认参数将 notes.xml 文件中的内容赋值给了变量$xmlObject。这时变量$xmlObject 是一个 SimpleXMLElement 对象。然后我们使用了该对象的 xpath()方法查找名为“note”的节点，并将所有“note”节点的内容存入数组$notes 中。接下来，用 foreach 语句遍历了数组$notes，输出了 notes.xml 文件中的内容。

值得注意的是 xpath()对象中使用的 xpath 表达式“//note”。这个表达式中使用的“//”指的是在所有节点中查找其后跟随的节点。这里“//note”意思就是在所有的节点中查找名为“note”的节点。关于更多 xPath 语法的内容，读者可以参考 w3school 提供的 XPath 教程。可以访问如下网址：

```
http://www.w3cschool.cn/index-14.html
```

例 10.12 和例 10.13 两段脚本虽然使用了不同的方法，但输出的结果却是如出一辙。这两段脚本输出的结果如图 10-6 所示。



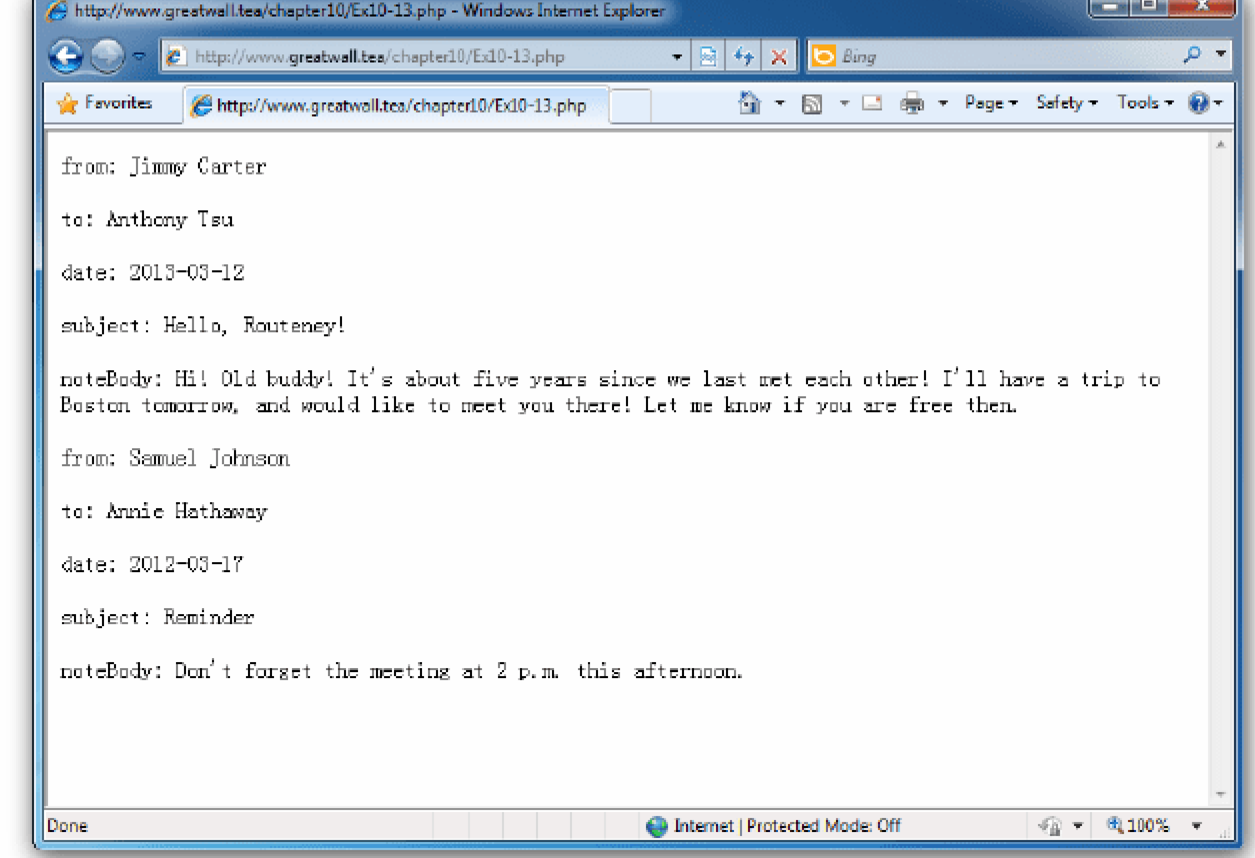


图 10-6　殊途同归的两段脚本

10.3　使用数据库存取数据

之前掌握了如何使用文本文件和 XML 文件来存取数据。但是我们发现用文本文件和 XML 文件来存取数据时，有一点不太方便，那就是删除数据。如果我们需要删除数据，就需要把所有的数据都读取出来放入若干的变量或者数组中，剔除掉与需要删除的数据相关的变量或数组元素，再把剩余的数据拼在一起以覆盖的方式写入原始文件中。操作起来十分地繁琐复杂。对于大量的数据来说，资源消耗将呈几何极数增长。

为了避免出现这种情况，可以使用关系型数据库来存取数据。通过使用 SQL 的 SELECT、CREATE、UPDATE 和 DELETE 等语句对数据库进行操作，使得增加、修改和删除指定的数据变得非常地方便。

在本节里，所有的内容都将围绕一个简单的图书管理系统展开介绍。我们可以使用这个图书管理系统来检查图书的存放和使用情况以及读者们对阅读图书后产生的评价。这个图书管理系统使用的是 MySQL 数据库，架构如图 10-7 所示。

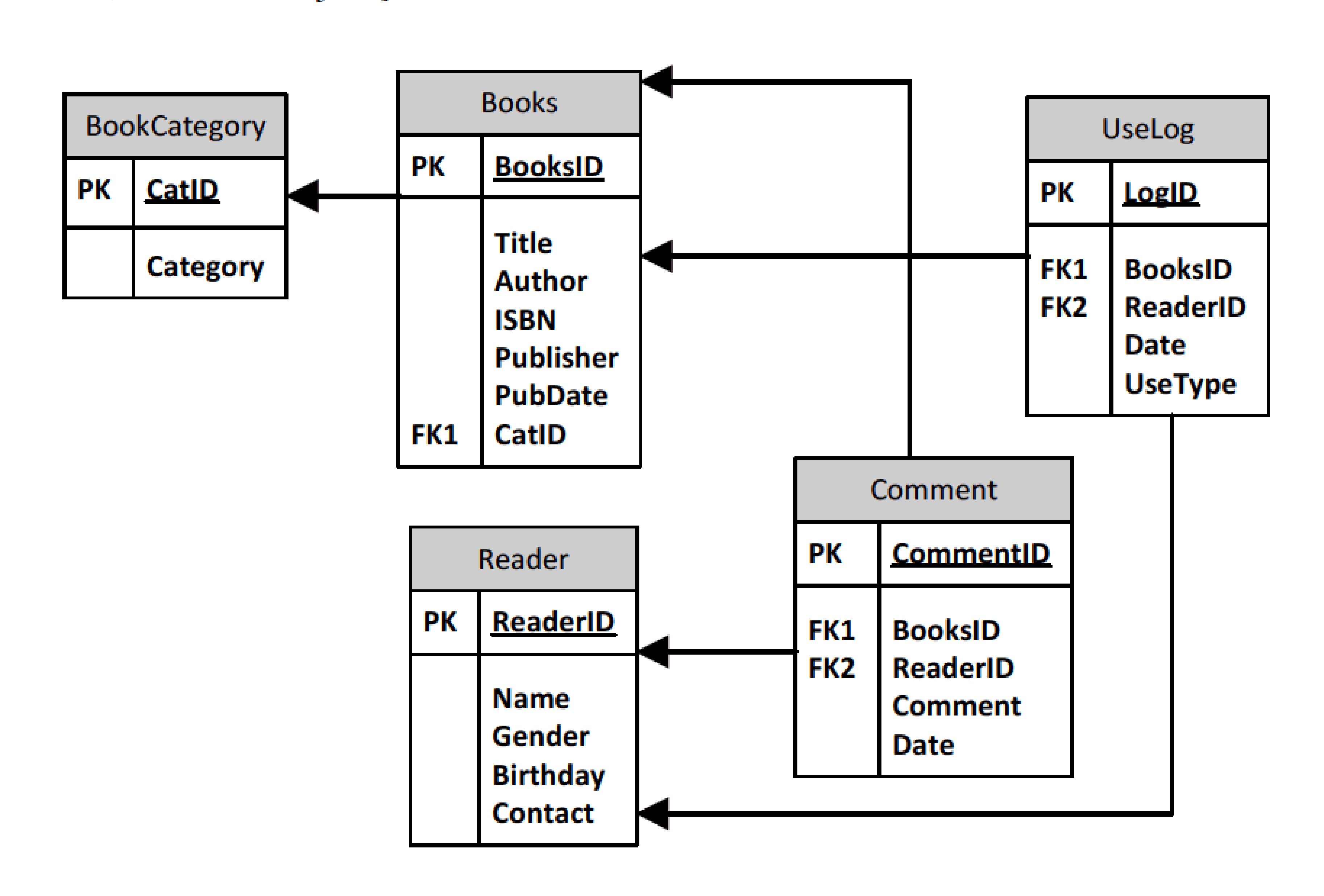


图 10-7　图书管理系统数据架构图

10.3.1　数据库基础

为了看懂如图 10-13 所示的数据架构图，我们需要掌握一些关于数据库的基础知识。

一个关系型数据库是由表以及表和表之间的关系组合起来的，这就是为什么这种类型的数据库称为关系型数据库。在我们这个数据库里一共有五张表，它们分别是：Books 表、Reader 表、UserLog 表、Comment 表和 BookCategory 表。表 10-3 所示描述了这些数据库表记录的内容：

表 10-3　图书管理系统中用到的数据库表

数 据 库 表	描　　述
Books	用于存放诸如书名、作者、出版日期等与图书有关的信息
Reader	用于存放诸如姓名、性别、出生年月、联系方式等与读者有关的信息
UserLog	用于存放诸如使用类型、借阅时限等与图书借阅行为有关的信息，该表与 Books 和 Reader 两张表之间存在着引用关系
Comment	用于存放读者在阅读其借阅的图书后产生的评价，该表与 Books 和 Reader 两张表之间存在着引用关系
BookCategory	用于给存放在 Books 表中的图书进行分类，该表与 Books 表之间存在着引用关系

通过表 10-3 所示，我们大致了解了这五张表的用途。现在就来看看如何在 MySQL 数据库服务器上创建这些数据库表吧。

如果你和我一样使用的是安装有 Windows 7 操作系统的计算机，并且将 MySQL 服务器软件安装在 C 盘，那么你可以在 C 盘下找到 MySQL 的安装文件夹，记录该文件夹的路径。我的路径是：

```
C:\Program Files\EasyPHP-12.1\mysql\
```

选择“开始”|“附件”|“命令提示符”命令，打开命令窗口。然后在当前路径下运行如下的命令：

```
cd \"Program Files"\EasyPHP-12.1\mysql\bin
```

进入 MySQL 安装路径下的 bin 文件夹。随后运行如下命令，以使用 root 用户登录 MySQL 服务器：

```
mysql -uroot
```

为了演示的方便，我们数据库 root 用户并没有加密。如果你在安装 MySQL 服务器的过程中为 root 用户指定了密码，那么请在上面的命令后添加“-p”参数，并写上密码。

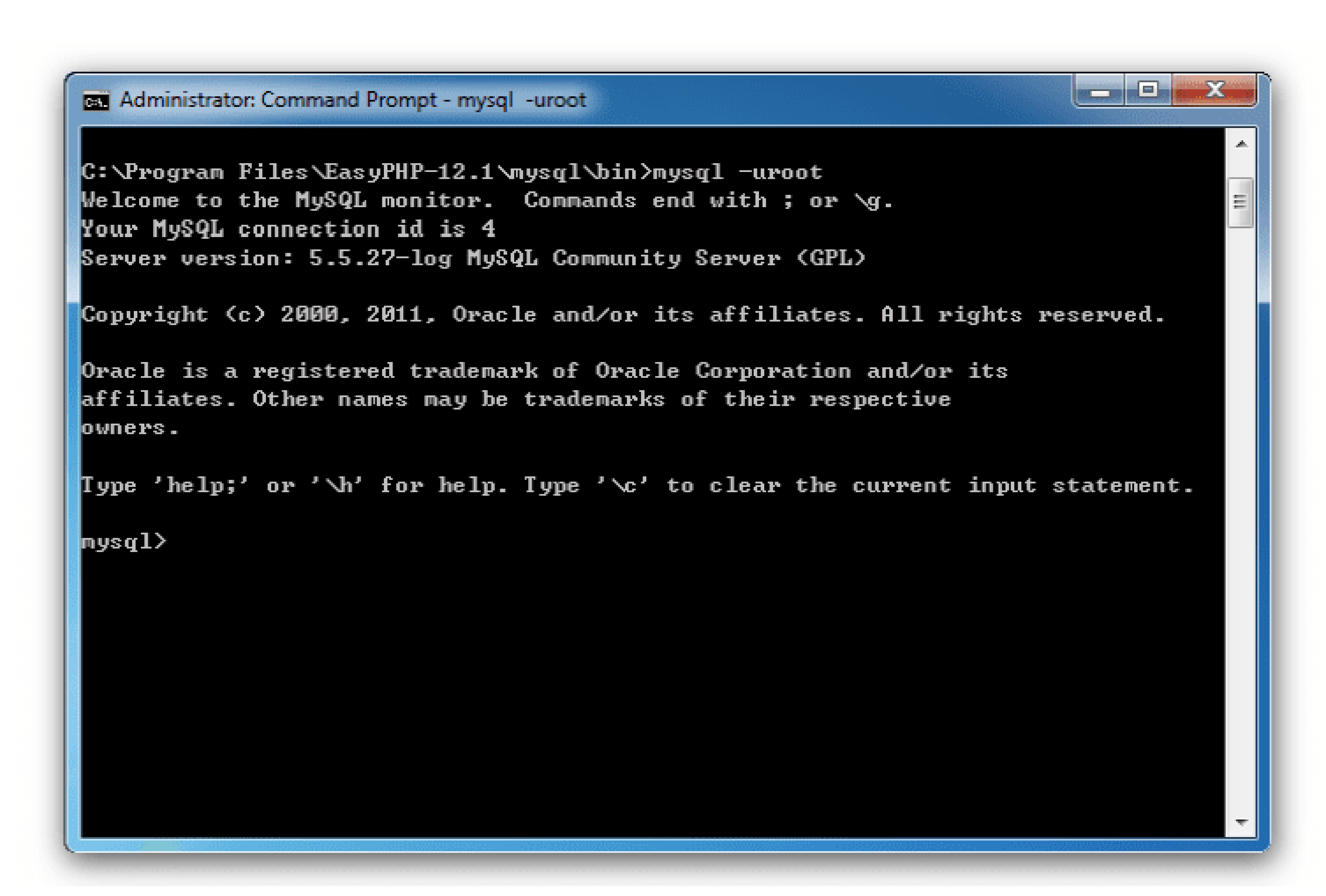


图 10-8　MySQL 命令行窗口

如果看到了如图 10-8 中所示的 MySQL 欢迎信息，那么恭喜你成功进入了 MySQL 命令行窗口。如果看到了如下所示的错误提示：

```
ERROR 2003 (HY000): Can't connect to MySQL server on 'localhost' (10061)
```

多半是由于 MySQL 数据库服务没有启动。只需要启动你的 MySQL 数据库服务即可。

如果成功启动了 MySQL 数据库服务，并且进入了 MySQL 命令行窗口，那就着手为这个图书管理系统创建数据库吧。

（1）创建数据库

在光标闪动的“mysql>”命令提示符后输入如下命令：

```
CREATE DATABASE LibraryManager;
```

按下回车键后，系统返回如下信息：

```
Query OK, 1 row affected (0.02sec)
```

表明，一个名为 LibraryManager 的数据库创建成功。

你可以使用如下的命令来查看当前 MySQL 服务器上的所有数据库：

```
SHOW DATABASES;
```

如果你可以在这条命令的回显信息中，找到我们刚才创建的数据库，那么数据库 LibraryManager 就创建成功了。

（2）创建数据库表

在创建数据库表之前，需要使用如下的命令指定一个数据库：

```
USE LibraryManager;
```

这时，系统回显“Database changed”，表明数据库切换成功。也就是说，我们可以创建数据库表了。

这里只演示如何创建数据库表 Books。其他的数据库表大家可以重复如下的步骤进行创建。

在光标闪动的“mysql>”后输入如下的命令：

```
CREATE TABLE books (
    bookid int NOT NULL AUTO INCREMENT PRIMARY KEY,
    title CHAR(50),
    author CHAR(50),
    ISBN CHAR(50),
    publisher CHAR(50),
    pubdate DATE NOT NULL,
    catid INT NOT NULL
) DEFAULT CHARACTER SET utf8 ENGINE=InnoDB;
```

这条命令中使用了 CREATE TABLE 关键字，告诉 MySQL 服务器我们要为当前数据库创建一张名为 Books 的表。括号里的内容是定义数据库表 Books 的结构。按照之前的规划，数据库表 Books 有七列，它们分别是 Bookid、Title、Author、ISBN、Publisher、Pubdate 和 Catid。每列可以容纳的数据类型不同，在定义数据库表时需要指定，比如 Bookid 和 Catid 两列只能容纳整数，Pubdate 列只能容纳时间，其余列只能容纳字符串。在命令的最后定义了数据表中存储的数据使用的字符集和数据引擎。在定义好一切后，按下回车键。当系统回显：

```
Query OK, 0 rows affected (0.19 sec)
```

表明数据库表 Books 已经创建成功。这时，可以使用 DESCRIBE 命令来打印数据库表 Books 的结构，如图 10-9 所示。

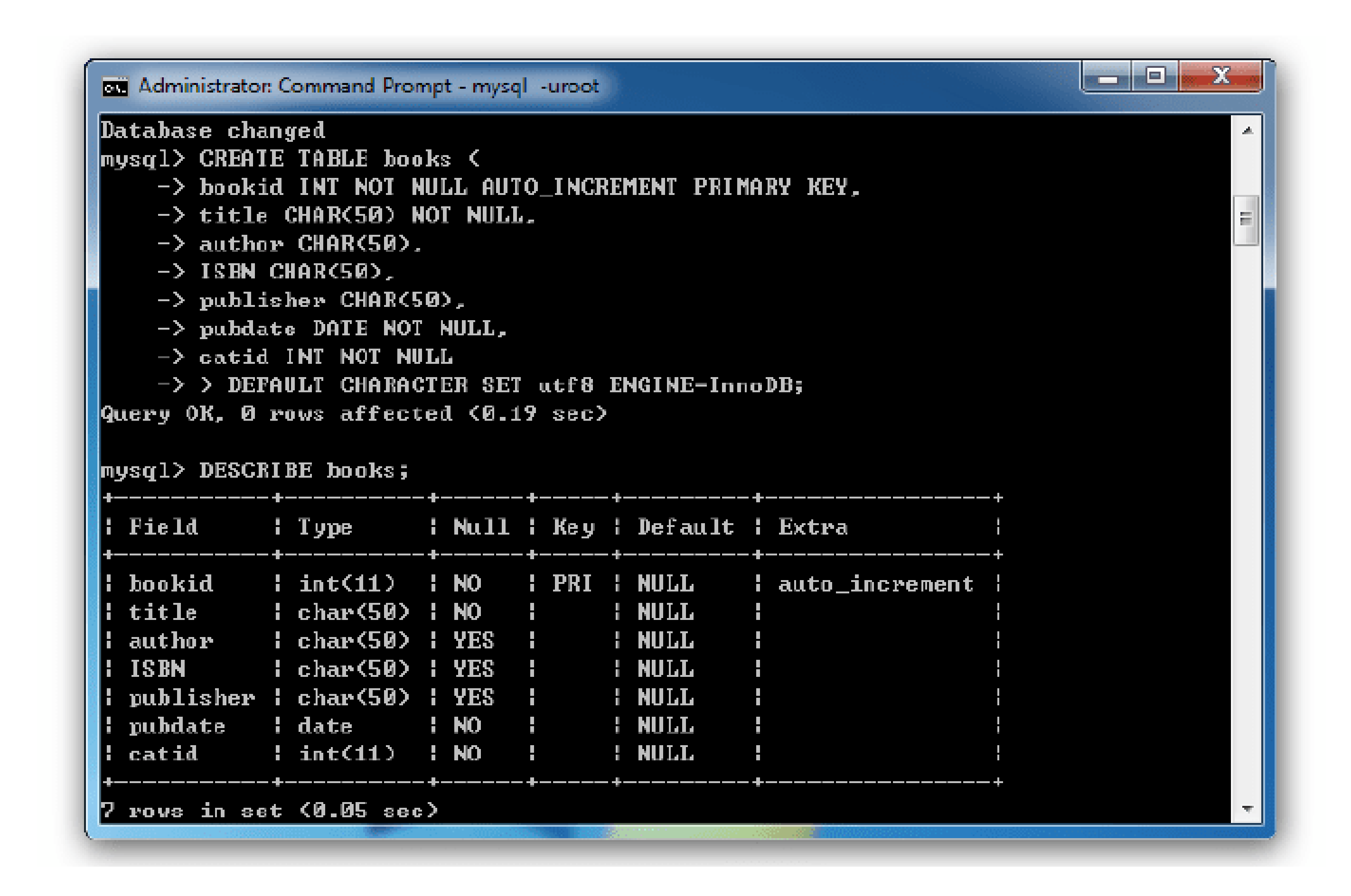


图 10-9　数据库表 books 的数据结构

在命令 DESCRIBE books 命令的回显信息中，我们看到了一张表。十分详尽地罗列了刚才定义的表格每一列的情况。值得注意的是，表格的第一列为该表的主键，不允许为空，其值会根据上一行的内容自动添加。

现在，我们可以按照之前的规划，把其他四张表添加到数据库中。添加完成后的结果如图 10-10 所示。

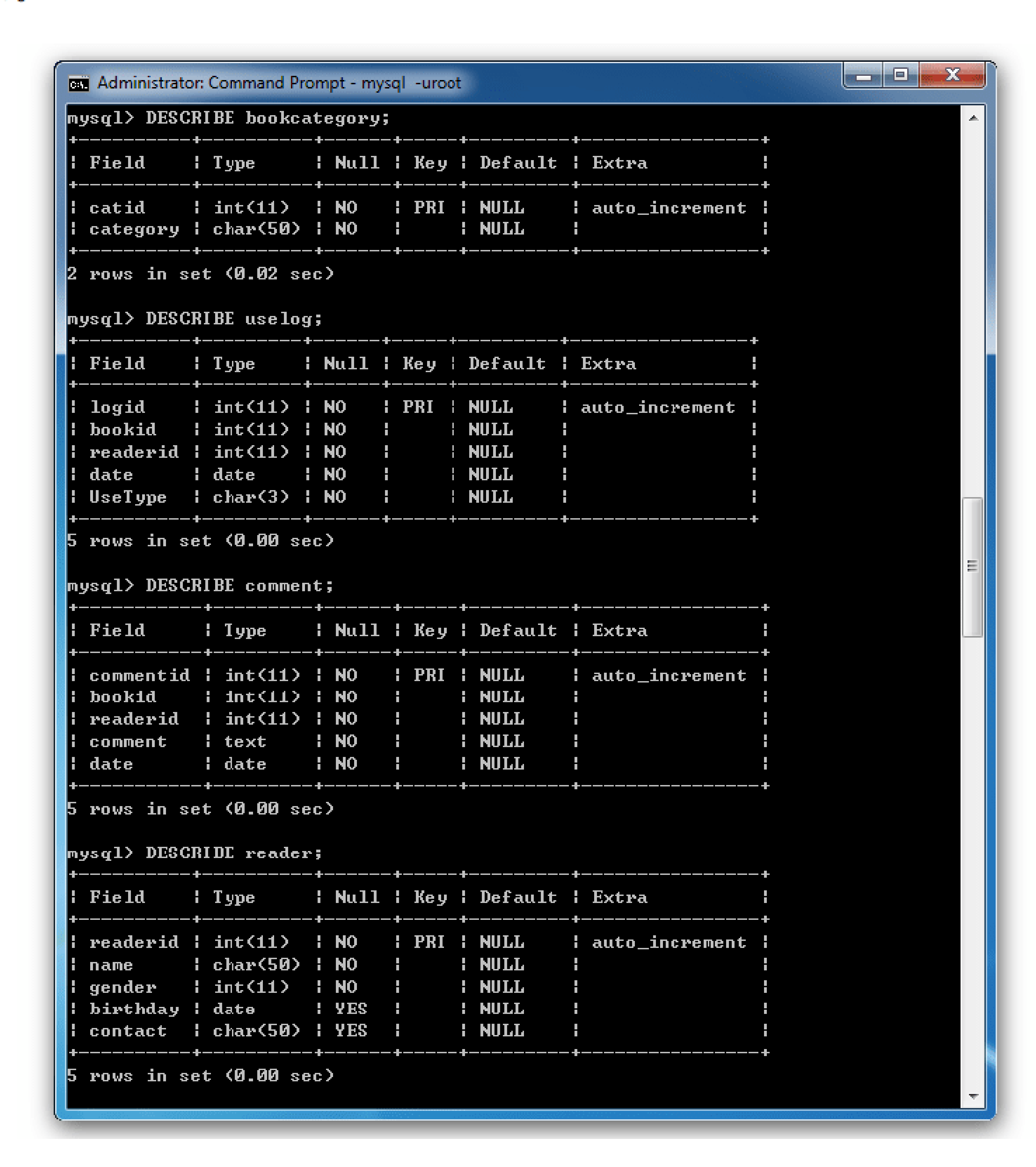


图 10-10　图书管理系统里的数据表

（3）向数据表中添加数据

在添加了所有计划的数据库表之后，我们就可以向数据库中添加数据了。先在 bookcategory 表中添加一条数据：

```
INSERT INTO bookcategory SET
    category='Literature';
```

这里使用了 INSERT INTO 关键字用来向 bookcategory 表中添加数据。之后使用 SET 关键字来设置 bookcategory 表中当前行与 category 列（或称为“字段”）相交的单元格的值为“Literature”。

这条命令也可以写成如下的样子：

```
INSERT INTO bookcategory
    (category) VALUES ('Computer Science');
```

如果需要同时指定多个字段的值，可以用逗号将它们隔开，就像下面这样：

```
INSERT INTO tableName
    (field1, field2, …) VALUES (value1, value2, …)
```

（4）查看数据表中存储的数据

无论数据表中是否存在数据，我们都可以使用 SELECT 语句来查看数据表的内容。只不过，如果数据表中没有数据，SELECT 指令返回的结果是：

```
Empty set (0.00 sec)
```

上一步中，我们在 bookcategory 表中添加了两条数据。现在，我们再查询一下 bookcategory 表中存储的数据：

```
SELECT * FROM bookcategory;
```

这条指令运行的结果是一张表格，列出了 bookcategory 表中现有的数据。其中，“*”做为通配符，代表所有的列。运行该指令后，一张两行两列的表格被打印了出来。其中第一列为“catid”，第二列为“category”。在上一步中，指定了 category 的值，并没有指定 catid 的值。但是“catid”具有 AUTO_INCREMENT 属性，所以系统会在上一条记录的值的基础上加 1 再走入当前记录中。若上一条记录不存在，则当前记录的值为 1。

我们也可以通过添加 WHERE 从句来显示需要的内容，比如：

```
SELECT * FROM bookcategory WHERE catid=2;
```

当指定了只显示 catid 为 2 的记录时，系统就只返回了数据表中的第二条记录。

如果只需要显示某条记录的某一列的值时，我们可以在 SELECT 关键字后指定该列，就像下面这样：

```
SELECT category FROM bookcategory WHERE catid=2;
```

这时，显示的表格除表头外只有一行一列，也就是 cateid 为 2 的那条记录的 category 字段的值。

（5）修改指定记录某字段的值

如果需要修改某条记录指定字段的值，我们可以使用 UPDATE 命令。UPDATE 命令的格式如下：

```
UPDATE tableName SET
    field1=value1,
    field2=value2,
    …
```

比如，我们想要把 bookcategory 表中的第二条记录的 category 字段的值从“Computer Science”修改为“Science”，就可以使用下面这条命令：

```
UPDATE bookcategory SET category='Science'
WHERE catid=2;
```

当系统返回如下信息时：

```
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed:1  Warnings: 0
```

指定记录的 category 字段的值修改成功。这时再使用 SELECT 命令显示 bookcategory 表格存储的所有数据时，就会发现新的数据替代了原有的数据。

（6）删除指定的记录

如果需要删除某条记录，我们可以使用 DELETE 命令。DELETE 命令的格式如下：

```
DELETE FROM tableName WHERE conditions
```

假如，我们需要删除 bookcategory 表中的 catid 为 2 的记录时，可以使用下面这条命令：

```
DELETE FROM bookcategory WHERE catid=2;
```

当系统返回如下信息时：

```
Query OK, 1 row affected (0.00 sec)
```

证明删除成功。这时我们再使用 SELECT 命令显示 bookcategory 表中的数据时，就看不到这条记录了。

10.3.2　数据表之间的关系

在数据库中，每张表独立存在，但又正如图 10-7 所示互相有着联系。当某张表格中的某条记录的某一个字段发生变化时，相关的表格中存在的数据也会发生变化。在本小节里，我们就讨论一下数据库中数据表之间的关系。

数据表之间存在的关系可以分为“一对一”和“一对多”两种关系[1]：

- 所谓“一对一”指的是一张表和另一张表之间存在着对应关系，比如图 10-7 所示中 Bookcategory 和 Books 两张表之间的关系就是一对一的关系，其中 Bookcategory 表的主键 CatID 做为 Books 表的外键让这两张表联系在了一起。
- 所谓“一对多”指的是一张表与多张表之间存在着的对应关系，比如图 10-7 所示中 Comment、Books 和 Reader 三张表之间存在着“一对多”的关系，其中 Books

按照通常的看法，应该还有一种“多对多”的数据表间关系。假设我们有表 A 和表 B 两张表，表 A 中的一条记录可以对应表 B 中的多条记录，同时表 B 中一条记录也可能对应表 A 中的多条记录。这样一来，两张表之间的关系就是“多对多”的。其实，我们可以把这个多对多的关系简化成两个一对多的关系。

表和 Reader 表的主键 BooksID 和 ReaderID 做为 Comment 表的外键将它们联系在了一起。

这里说到主键和外键的概念，也顺便提一句。所谓主键指的是一张数据表中用来区分一条记录与另一条记录的字段，同一张表中的每个记录的主键值都必须不同。通常情况下，我们会使用“编号”来做一张数据表的主键。所谓外键，就是指一张数据表中与其他数据表中的主键相同的非主键字段，它是用来联络数据表之间的关系的。

在本节里，我们就从以两种表间关系为例来讨论一下数据表之间的关系。

1. 一对一的表间关系

在讨论一对一的表间关系之前，先在 Bookcategory 和 Books 两张表中添加一些数据。具体请参见图 10-11 所示的内容。

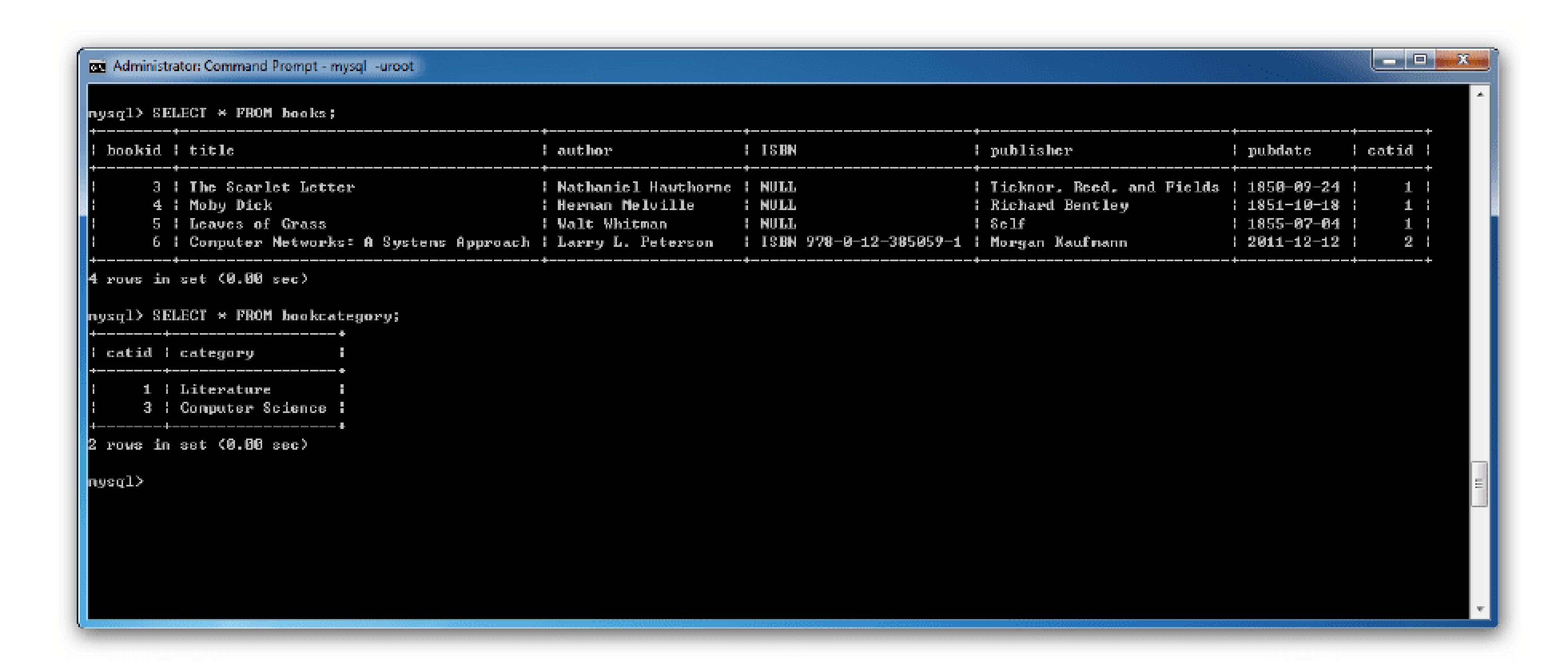


图 10-11　一对一的表间关系

现在，Books 表中有 4 条记录，而 Bookcategory 表中有 2 条记录。两张表之间存在着一对一的关系，因为 Bookcategory 表的主键 CatID 是 Books 表的外键，而 Books 表中有且仅有一个外键。现在可以把两张表的数据结合起来，输出我们想要输出的部分。比如想查找 Books 数据表中由 Self 出版社出版的图书所属的图书类型，那么就可以执行如下的命令：

```
SELECT books.title, books.author, bookcategory.category FROM books, bookcategory
WHERE books.catid = bookcategory.catid AND
books.publisher = 'Self';
```

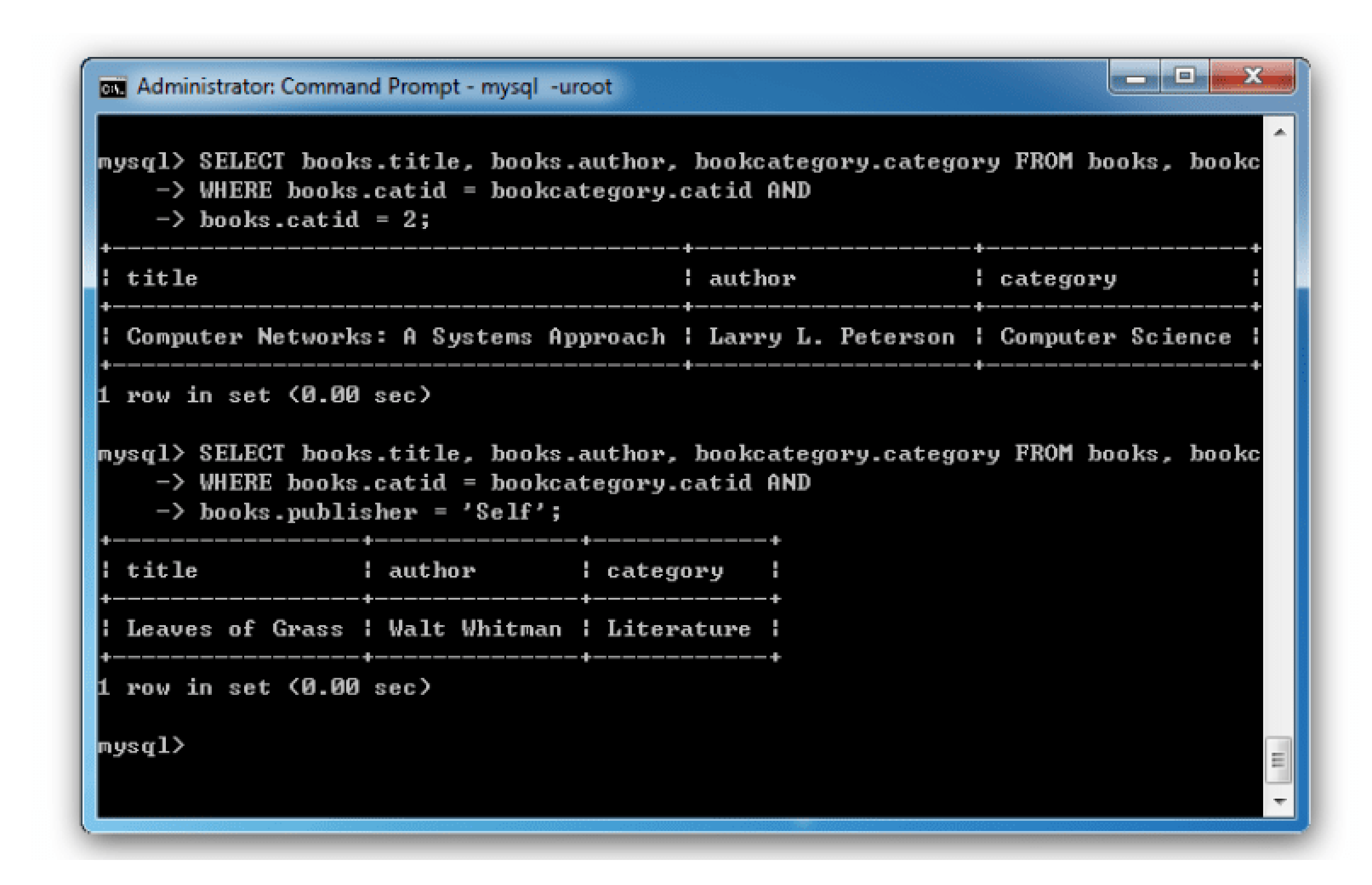
再比如，我们想要查找 Books 数据表中图书类型为“Computer Science”的图书列表，那么就可以执行如下的命令：

```
SELECT books.title, books,author, bookcategory.category FROM books, bookcategory
WHERE books.catid = bookcategory.catid AND
books.catid = 2;
```

在这两条命令中，我们使用 SELECT 分别从两张表中读取了若干字段，然后使用 WHERE 从句指定了两表之间的关系和查找条件。值得我们注意的是：

- ❑ SELECT 关键字后面跟随的字段名称前面都带上了表名；
- ❑ 指定两张表之间的关系时，我们用等号将两张表里相同的字段连接在了一起；
- ❑ AND 是逻辑运算符，表示只有当前后两个条件都满足时，才会显示结果。

上面这两条命令的结果如图 10-12 所示。



```
Administrator: Command Prompt - mysql -uroot
mysql> SELECT books.title, books.author, bookcategory.category FROM books, bookc
    -> WHERE books.catid = bookcategory.catid AND
    -> books.catid = 2;
+-------------------------------------+-------------------+------------------+
| title                               | author            | category         |
+-------------------------------------+-------------------+------------------+
| Computer Networks: A Systems Approach | Larry L. Peterson | Computer Science |
+-------------------------------------+-------------------+------------------+
1 row in set (0.00 sec)

mysql> SELECT books.title, books.author, bookcategory.category FROM books, bookc
    -> WHERE books.catid = bookcategory.catid AND
    -> books.publisher = 'Self';
+-----------------+--------------+------------+
| title           | author       | category   |
+-----------------+--------------+------------+
| Leaves of Grass | Walt Whitman | Literature |
+-----------------+--------------+------------+
1 row in set (0.00 sec)

mysql>
```

图 10-12　基于一对一的表间关系进行选择的结果

2．一对多的表间关系

在讨论一对多的表间关系之前，先在 Reader 表和 Uselog 表中添加一些数据，如图 10-13 所示。

```
Administrator: Command Prompt - mysql -uroot
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM uselog;
+-------+--------+----------+------------+---------+
| logid | bookid | readerid | date       | UseType |
+-------+--------+----------+------------+---------+
|     1 |      3 |        1 | 2013-03-24 | BOR     |
|     2 |      5 |        4 | 2013-03-24 | BOR     |
|     3 |      6 |        1 | 2013-03-24 | RET     |
+-------+--------+----------+------------+---------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM reader;
+----------+-----------------+--------+------------+-------------------+
| readerid | name            | gender | birthday   | contact           |
+----------+-----------------+--------+------------+-------------------+
|        1 | James Bond      |      0 | 1975-02-14 | 0086-010-12458956 |
|        2 | Marilyn Monroe  |      1 | 1965-06-01 | 0086-020-56984575 |
|        3 | Kaitlyn Jenkins |      1 | 1982-01-25 | 0086-021-45897569 |
|        4 | Judy Garland    |      1 | 1992-10-12 | 0086-027-45698956 |
+----------+-----------------+--------+------------+-------------------+
4 rows in set (0.00 sec)

mysql>
```

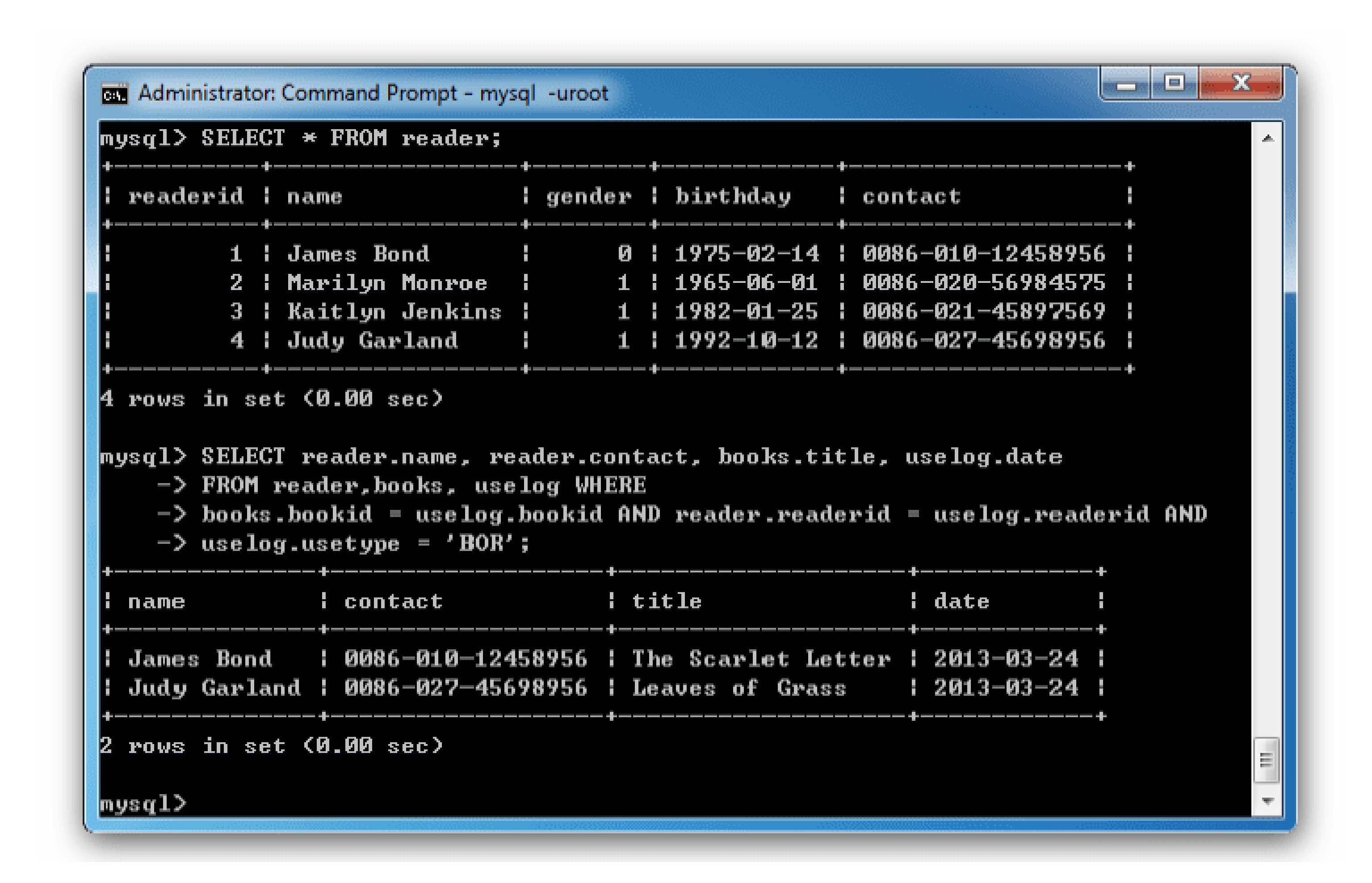
图 10-13　一对多的表间关系

现在 Books 表中有 4 条数据、Reader 表中有 4 条数据，而 Uselog 表中有 3 条数据。在 Uselog 表中，usetype 列的“BOR”代表的是“借出”，而“RET”代表的是“归还”。

假如，我们需要查找所有已借出图书及读者的相关信息，可以运行如下的命令：

```
SELECT reader.name, reader.contact, books.title, uselog.date
FROM reader, books, uselog WHERE
books.bookid = uselog.bookid AND reader.readerid = uselog.readerid AND
uselog.usetyp = 'BOR';
```

运行该命令后，系统会返回所有已借图书及读者的相关信息，如图 10-14 所示。


```
mysql> SELECT * FROM reader;
+----------+-----------------+--------+------------+------------------+
| readerid | name            | gender | birthday   | contact          |
+----------+-----------------+--------+------------+------------------+
|        1 | James Bond      |      0 | 1975-02-14 | 0086-010-12458956 |
|        2 | Marilyn Monroe  |      1 | 1965-06-01 | 0086-020-56984575 |
|        3 | Kaitlyn Jenkins |      1 | 1982-01-25 | 0086-021-45897569 |
|        4 | Judy Garland    |      1 | 1992-10-12 | 0086-027-45698956 |
+----------+-----------------+--------+------------+------------------+
4 rows in set (0.00 sec)

mysql> SELECT reader.name, reader.contact, books.title, uselog.date
    -> FROM reader,books, uselog WHERE
    -> books.bookid = uselog.bookid AND reader.readerid = uselog.readerid AND
    -> uselog.usetype = 'BOR';
+--------------+-------------------+-------------------+------------+
| name         | contact           | title             | date       |
+--------------+-------------------+-------------------+------------+
| James Bond   | 0086-010-12458956 | The Scarlet Letter | 2013-03-24 |
| Judy Garland | 0086-027-45698956 | Leaves of Grass   | 2013-03-24 |
+--------------+-------------------+-------------------+------------+
2 rows in set (0.00 sec)

mysql>
```

图 10-14　基于一对多的表间关系进行选择的结果

10.3.3　查询结果的排序和组合

有时，我们也需要对数据进行排序和组合，以方便数据的呈现和统计。现在就以 Reader 表中的数据为例，来看看如何使用 ORDER BY 从句和 GROUP BY 从句。

1．查询结果的排序

如果我们想要让 Reader 表中的数据按照读者的生日从大到小进行排序，那么，就可以运行如下的命令：

```
SELECT * FROM reader ORDER BY birthday;
```

运行这条命令后，会发现 readerid 为 2 的读者 Marilyn Monroe 被放到了第一行，而 readerid 为 1 的读者 James Bond 被放到了第二行。

默认情况下，ORDER BY 子句会按照从大到小（数量上），从前往后（时间上）的顺序进行排序，使用 ASC 关键字。比如上面那条命令还可以写成：

```
SELECT * FROM reader ORDER BY birthday ASC;
```

如果想让查询的结果按照从小到大（数量上），从后往前（时间上）的顺序进行排序的话，就得使用 DESC 关键字了。比如：

```
SELECT * FROM reader ORDER BY birthday DESC;
```

这时，排在查询结果最前面的一条记录是 readerid 为 4 的 Juday Garland。而 readerid 为 2 的 Marilyn Monroe 则排在了最后。

2．查询结果的分组

如果想统计 Books 数据表中类型为“Literature”和“Computer Science”的图书各有多

少本，我们可以运行如下的命令：

```
SELECT bookcategory.category, COUNT(books.title) AS count FROM books,
bookcategory
WHERE books.catid = bookcategory.catid
GROUP BY books.catid;
```

在这条命令中使用了 COUNT()函数、AS 关键字和 GROUP BY 从句，其中 COUNT()函数的功能是统计符合条件的记录一共有多少条、AS 关键字可以将它前面的字段名或表达式改个名字，而 GROUP BY 从句则指定排序的标准。运行这条命令后，系统返回了一张表格，分别列出了类型为“Literature”和“Computer Science”的图书各有多少本，如图 10-15 所示。

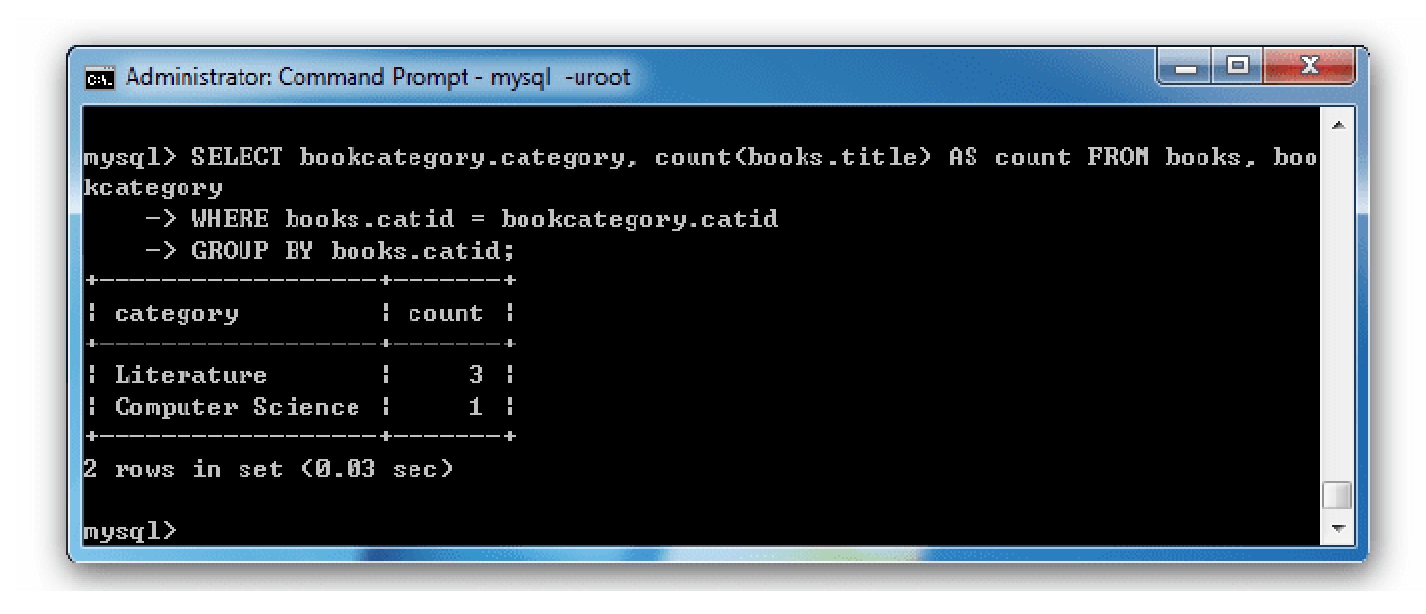


图 10-15　查询结果的分组

除了 COUNT()函数之外，可以和 GROUP BY 子句搭配使用的函数还有：AVG()、MIN()、MAX()和 SUM()。它们的功能分别是计算查询结果的平均值、最小值、最大值和总和。

10.4　使用 PHP 来操作数据库

在上一节里通过 MySQL 的命令行窗口学习了简单的 SQL 命令，知道如何使用 SQL 命令创建数据库和数据表、向数据表中添加数据以及在数据表中查找符合指定条件的记录。其实，我们可以通过编写 PHP 脚本来实现这一切。

不知道大家是否还记得，在第 2 章里讲了如何安装 phpMyAdmin 这款软件。图 10-16 展示了以 root 用户登录 phpMyAdmin 的显示的管理界面。

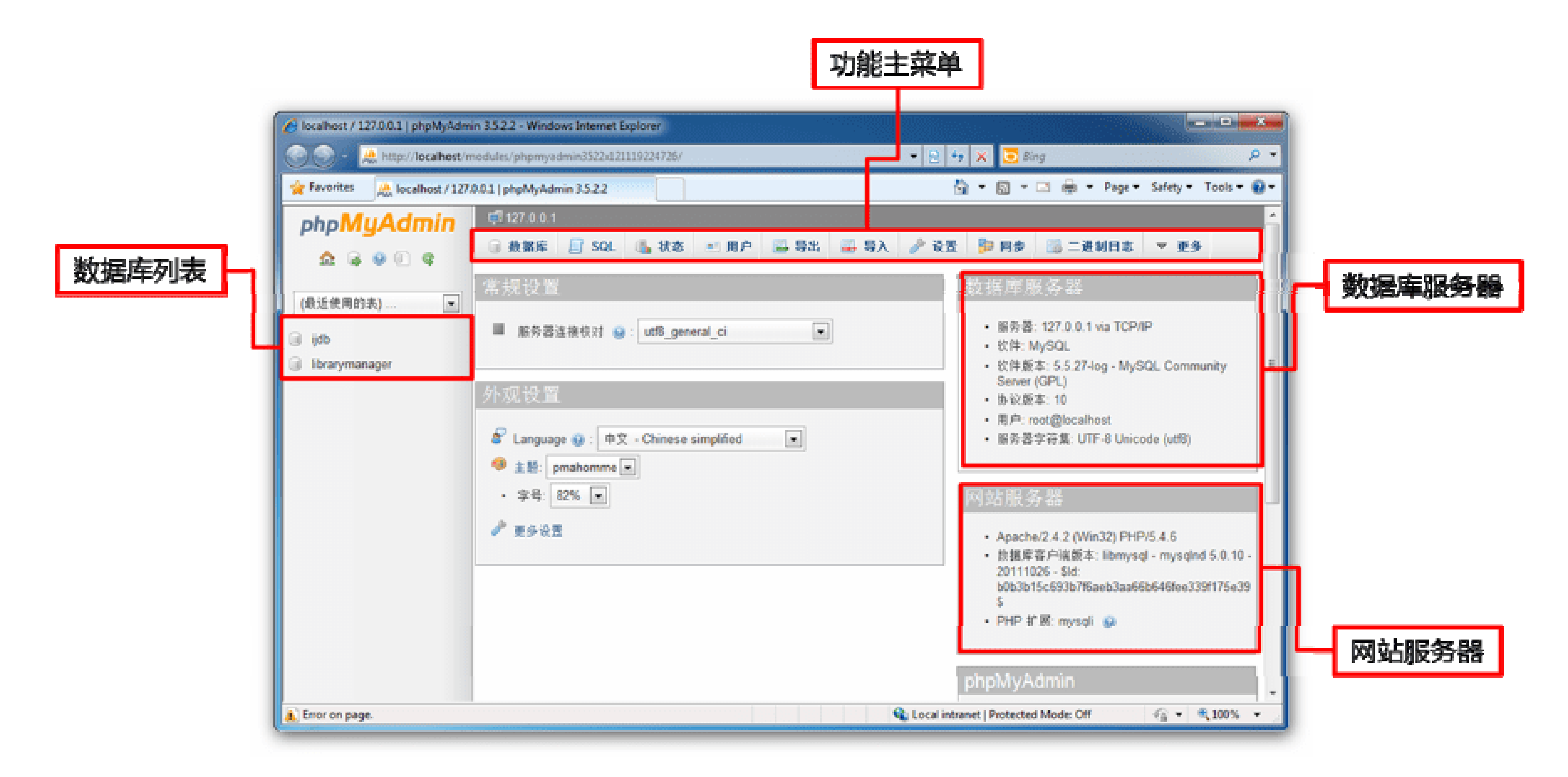


图 10-16　phpMyAdmin 管理界面

使用 phpMyAdmin 可以很方便地对数据库进行创建数据表、添加数据、修改数据、删除数据和查找数据等一系列操作。而所有这一切都是 PHP 脚本的功劳。

PHP 为我们提供了三套应用程序接口（API）来和 MySQL 数据库进行配合，它们分别是 MySQLi、PDO 和 MySQL。PHP 官方推荐我们使用前两种。按照官方的说法，MySQL（API）已经处于 Deprecated 状态，也就是不再开发和维护了。表 10-4 所示比较了这三种应用程序接口的特点。

表 10-4　MySQL数据库扩展特性比较

	ext/mysqli	PDO_MySQL	ext/mysql
开始支持版本	5	5.1	2
MySQL 5.x 是否提供支持	是	是	是
开发状态	活跃	活跃	仅维护
生命周期	活跃	活跃	已停止
新开发项目是否推荐	是	是	否
基于对象编程接口	是	是	否
基于过程编程接口	是	否	是
支持使用 mysqlnd 执行非块状异步查询	是	否	否
长连接	是	是	是
支持字符集	是	是	是
支持数据库事务	是	是	否

从表 10-4 中可以看出在 PHP 5 之前，广泛使用的是 PHP 的 MySQL 扩展对 MySQL 数据库进行操作的，该扩展因不支持基于对象编程而被废弃。在 PHP 5 发布时，PHP 为其增加了一项过渡性质的扩展，即 mysqli，该扩展同时支持基于对象编程和基于过程编程。在发布 PHP 5 的第一个子版本时，PHP 又推出了一项功能更加全面的数据库操作接口，名为 PHP Data Objects（PDO），它仅支持基于对象编程，将包括 MySQL 数据库在内的所有数据库类的操作都整合在了一起。

在下面的例程中，我们将使用 MySQLi 和 PDO 两种方式进行脚本的编写。在脚本的编写中尽量突出两种扩展基于对象的特点。

好了，闲话少说，我们来看看如何使用 PHP 操作 MySQL 数据库吧。在此之前，先为 LibraryManager 数据库分配一个管理员“libarian”，其安全密钥为“librarian9235”。

```
GRANT ALL PRIVILEGES ON librarymanager.*
TO librarian@localhost IDENTIFIED BY 'librarian9235'
WITH GRANT OPTION;
```

在这条指令中：

- GRANT 表示进行权限分配，ALL PRIVILEGES 表示所有权限，也可以指定具体的权限，如 SELECT、UPDATE 和 CREATE 等。具体可见表 10-5 详细查看 MySQL 用户常用权限。
- ON 后跟随的是指定的数据库以及数据库中指定的表格。在上例中，我们指定的 librarymanager 数据库中的所有表格。

- TO 后跟随的是数据库用户名，注意在用户名后要加“@”和主机名。
- IDENTIFIED BY 后跟随的是数据库用户的密码，也就是之前提到的安全密钥。
- WITH GRANT OPTION 表示该用户可以分配权限给其他用户。

在指定了这些参数之后，我们就可以创建一个 MySQL 用户，并为其分配可管理的数据库和相关管理权限。

表 10-5　MySQL用户常用权限

权　　限	列	上　下　文
CREATE	Create_priv	数据库、表或索引
DROP	Drop_priv	数据库或表
GRANT OPTION	Grant_priv	数据库、表或保存的程序
REFERENCES	References_priv	数据库或表
ALTER	Alter_priv	表
DELETE	Delete_priv	表
INDEX	Index_priv	表
INSERT	Insert_priv	表
SELECT	Select_priv	表
UPDATE	Update_priv	表
CREATE VIEW	Create_view_priv	视图
SHOW VIEW	Show_view_priv	视图
ALTER ROUTINE	Alter_routine_priv	保存的程序
CREATE ROUTINE	Create_routine_priv	保存的程序
EXECUTE	Execute_priv	保存的程序
FILE	File_priv	服务器主机上的文件访问
CREATE TEMPORARY TABLES	Create_tmp_table_priv	服务器管理
LOCK TABLES	Lock_tables_priv	服务器管理
CREATE USER	Create_user_priv	服务器管理
PROCESS	Process_priv	服务器管理
RELOAD	Reload_priv	服务器管理
REPLICATION CLIENT	Repl_client_priv	服务器管理
REPLICATION SLAVE	Repl_slave_priv	服务器管理
SHOW DATABASES	Show_db_priv	服务器管理
SHUTDOWN	Shutdown_priv	服务器管理
SUPER	Super_priv	服务器管理

更多关于数据库用户的操作指令，读者可以参考 PHPKnowHow 网站上的“Managing MySQL Users”一文。这篇文章的 URL 是 http://www.phpknowhow.com/mysql/managing-mysql-users/。

10.4.1　使用 PHP 打开和关闭数据库连接

前面提到过 MySQLi 扩展同时支持基于对象开发和基于过程开发。那么体现在连接数据库上，MySQLi 扩展提供了两种连接数据库的方式。其中，一种是基于过程的数据库连接，而另一种则是基于对象的数据库连接。

基于过程的数据库连接：

```
$mysql = mysqli_connect($hostname, $myuser, $mypassword, $mydb);
```

基于对象的数据库连接：

```
$mysql = new mysql($hostname, $myuser, $mypassword, $mydb);
```

注意如果在安装 MySQL 时修改了 MySQL 服务端口。那么在这两条命令中，还得提供第五个参数，MySQL 服务端口号。

如果我们使用 PDO 扩展，可以使用类似下面的指令连接数据库：

```
$pdoMySQL = new PDO('mysql:host=hostname;dbname=mydb', $myuser, $mypass);
```

创建一个 PDO 对象需要三个参数：第一个参数指定了主机名和数据库名，称为数据源字符串，第二个参数指定了数据库的管理员名，第三个参数指定了数据库管理员的密码。

【例 10.14】 使用 MySQLi 和 PDO 扩展连接 MySQL 数据库。

```
<?php
    // 使用基于过程的编程方式
    $conn = @mysqli_connect('localhost', 'librarian', 'librarian9235',
    'librarymanager');

    if (mysqli_connect_errno($conn))
        echo 'Failed to connect to MySQL: '.mysqli_connect_error();


    // 使用基于对象的编程方式
    $conn = @new mysqli('localhost', 'librarian', 'librarian9235',
    'librarymanager');

    if ($conn->connect_errno)
        echo 'Failed to connect to MySQL: '.$conn->connect_error;


    // 使用 PDO_MySQL 的连接方式
    try {
        $conn = @new pdo('mysql:host=localhost;dbname=librarymanager',
        'librarian', 'librarian9235');
    } catch (PDOException $err) {
        echo $err->getMessage();
    }
?>
```

在上面的三段脚本中，我们发现在定义数据库连接时，无论是使用 MySQLi 扩展的基于过程的编程、MySQLi 扩展的基于对象的编程、还是 PDO 扩展，都在定义连接的语句前使用了“@”符号。添加这个符号后，若该句存在错误，则可以避免系统输出该错误。这样一来，我们就可以自主的确定何时，在何处显示错误提示了。

在使用 MySQLi 扩展的基于过程的编程中，使用了 MySQLi 扩展为我们提供的独立的功能函数 mysqli_connect_errno()、mysqli_connect_error()和 mysqli_connect()。在这三个函数中，前两个函数各返回一个字符串，而后一个函数返回一个 mysqli 对象。

在使用 MySQLi 扩展的基于对象的编程中，我们把与 MySQL 数据库服务器的连接直接定义成了一个 mysqli 对象，然后使用了该对象的 connect_errno 和 connect_error 属性来判断连接是否成功。

而对于 PDO 扩展，我们使用了 try…catch 结构和 PDOException 对象来判断数据库连接是否正常。其中 PDOException 对象是 Exception 对象的扩展。

知道了如何连接指定的数据库，现在再来看看如何关闭这个数据库连接。虽然 PHP 会在当前页面所有的脚本执行完毕后，自动关闭数据库连接。但是 PHP 的 MySQLi 扩展基于过程的编程方法还是为我们提供了一个用于关闭当前数据库连接的方法 mysql_close($dbConn)。建议大家养成良好的习惯，在不需要数据库连接时使用 mysql_close($dbConn)方法关闭该连接。这个方法只有一个参数，那就是已建立的数据库连接。如果使用的 MySQLi 基于对象的编程方法，那么可以使用 mysqli 对象的 close()方法来关闭指定的数据库连接。对于使用 PDO 扩展创建的连接，我们只需要将存储该连接的变量设置为空即可断开该连接。

10.4.2　使用 PHP 输出数据库查询结果

在这一步里，我们还是用三种方式分别来尝试一下如何输出查询结果。

【例 10.15-1】 使用 MySQLi 扩展基于过程的编程输出查询结果。

```
<?php
    // 和 MySQL 数据库服务器建立连接
    $dbConn = @mysqli_connect('localhost', 'librarian', 'librarian9235',
    'librarymanager');
    if (mysqli_connect_errno($dbConn))
        die('Failed to connect to the database: '.mysqli_connect_
        error($dbConn));

    // 将查询指令存入$query 变量中
    $query = "SELECT bookid, title, author FROM books";

    //获取 MySQLi 查询结果，并将结果存入$result 变量中，该变量为 MySQLi 结果对象
    $result = mysqli_query($dbConn, $query);

    //使用 mysqli_fetch_array()方法指定的 MySQLi 结果对象转换成数组，并遍历数组
    while($book = mysqli_fetch_array($result)){
        echo 'ID '.$book[0].' <i>'.$book[1].'</i> by '.$book[2];
        echo '<br>';
    }

    // 后续处理
    mysqli_free_result($result);
    mysqli_close($dbConn);
?>
```

在例 10.15-1 的这段脚本中，我们使用了基于过程的编程。在编写脚本的过程中，使用了 MySQLi 扩展为我们提供的 mysqli_query()、mysqli_fetch_array()、mysqli_free_result()和 mysqli_close()独立函数。它们的作用如下所示。

- mysqli_query()方法用来执行 SQL 查询指令：该方法的第一个参数为字符中变量，返回的结果为一个 MySQLi 结果对象。
- mysqli_fetch_array()方法用来读取 MySQLi 结果对象中存储的数据：该方法的第一个参数为 MySQLi 结果对象，返回的结果为一个数组。
- mysqli_free_result()方法用来释放 MySQLi 结果对象占用的内存：该方法的第一个

参数为 MySQLi 结果对象，返回的结果为空值（NULL）。

- mysqli_close()方法用来断开和 MySQL 数据库的连接：该方法的第一个参数为 MySQLi 数据库连接对象，若连接断开，返回的结果为 TRUE。

我们也可以使用 MySQLi 基于对象的编程来编写功能相同的脚本。

【例 10.15-2】 使用 MySQLi 扩展基于对象的编程输出查询结果。

```
<?php
    // Establish the MySQL connection
    $dbConn = @new mysqli('localhost', 'librarian', 'librarian9235',
    'librarymanager');
    if($dbConn->connect_errno)
        die('Failed to connect to the database: '.$dbConn->connect_error);

    // Prepare the SQL statement
    $query = 'SELECT bookid, title, author FROM books';

    // Prepare the result for the mysql_fetch_array() method
    $result = $dbConn->query($query);

    // Iterate $result
    while($book = $result->fetch_array()){
        echo 'ID '.$book[0].' <i>'.$book[1].'</i> by '.$book[2];
        echo '<br>';
    }

    // Post-process
    $result->free();
    $dbConn->close();
?>
```

在这段脚本中，我们使用了基于对象的编程方法，把$dbConn 定义成一个 mysqli 对象。这个对象有 query()和 close()方法以及 connect_errno()和 connect_error()属性。其中 query()方法返回的值是个 mysqli 结果对象，而这个对象则拥有 fetch_array()和 free()方法。

现在再使用 PDO 扩展来看看是不是可能得到同样的结果。

【例 10.15-3】 使用 PDO 扩展输出查询结果。

```
<?php
    try {
        $dbConn = @new PDO('mysql:host=localhost;dbname=
        librarymanager','librarian','librarian9235');

        $query = 'SELECT bookid, title, author FROM books';

        foreach ($dbConn->query($query) as $book) {
            echo 'ID '.$book[0].' <i>'.$book[1].'</i> by '.$book[2];
            echo '<br>';
        }
    } catch (PDOException $err) {
        echo $err->getMessage();
    }
?>
```

运行了这三段程序后，会发现得到的结果都是一样的：

```
ID 3 The Scarlet Letter by Nathaniel Hawthorne
ID 4 Moby Dick by Herman Melville
ID 5 Leaves of Grass by Walt Whitman
```

```
ID 6 Computer Networks: A Systems Approach by Larry L. Peterson
```

需要注意的是，在使用 PDO 扩展时，PHP 并没有在 PDO 扩展中为我们提供与 MySQLi 扩展对应的对象和方法，而是提供了一套自有的方法。而这些方法通常比 MySQLi 扩展更加高效。

关于 MySQLi 扩展提供的对象及其属性和方法，PHP 官方网站上提供了一张完整的对照表格，读者可以在这张表格上找到 MySQLi 对象相同属性和具有相同功能的方法在基于对象和基于过程编程时的不同表现。具体的表格可以在 http://www.php.net/manual/zh/mysqli.summary.php 页面上找到。

关于 PDO 扩展提供的对象及其属性和方法，读者也可以参考 PHP 官方网站上的为我们提供的相关说明。具体的网址是 http://www.php.net/manual/en/book.pdo.php。

10.4.3　使用 PHP 来添加、修改和删除数据库数据

在第三节里，我们学习了如何执行 SQL 语句来添加修改和删除数据库中的数据。现在来看看如何通过 PHP 来执行 SQL 语句。

PHP 的 MySQLi 和 PDO 扩展都支持预编译语句（Prepared Statement）的使用。什么是预编译语句呢？所谓的预编译语句指的就是一个执行 SQL 指令的流程。在这个流程中，我们可以预先编译一条 SQL 指令，其中若干参数的值没有指定，接着绑定若干变量到这些参数，最后正式执行这条语句。这样一来，语句的复杂程度和数量都有所提升。那么这么做有什么好处呢？

按照通常的认识，使用预编译语句可以带来如下好处：

- 提高代码的可读性和可维护性；
- 提高代码执行的性能；
- 提高了代码执行的安全性。

因此，大家在处理对数据库的写操作时，务必使用预编译。这样即可以保证存储在数据库中的数据安全，还可以实现上述好处。

现在，我们一起来看看如何使用 PHP 来添加、修改和删除数据库中的记录吧。

首先，我们使用 MySQLi 扩展面向对象的编程方法在数据表 Bookcategory 中添加和删除一条记录。

【例 10.16-1】使用 MySQLi 扩展面向对象的编程方法在指定数据表中添加和删除一条记录。

```
<?php
    $dbConn = @new mysqli('localhost','librarian','librarian9235',
    'librarymanager');
    if ($dbConn->connect error) {
        die('Failed to connect to the database: '.$dbConn->connect error);
    }

    //指定图书类目
    $category = 'Productivity';

    //准备 SQL 查询语句
    if($stmt = $dbConn->prepare('SELECT * FROM bookcategory WHERE
```

```
    category=?')){

        //将指定的图书类目绑定到准备好的 SQL 查询语句中
        $stmt -> bind_param('s', $category);

        //执行 SQL 查询语句
        $stmt -> execute();

        //将查询结果绑定到指定变量中
        $stmt -> bind_result($sCatID, $sCat);

        //获取查询结果
        $stmt -> fetch();

        //关闭当前查询
        $stmt -> close();

        /*检查指定图书类目是否存在。若存在，删除该类目。
          若不存在，添加该类目 */
        if (!isset($sCatID)){
            //准备添加指定图书类目
            If($stmt = $dbConn->prepare('INSERT INTO bookcategory SET
            category=?')){

                //将指定图书类目绑定到准备好的 SQL 语句中
                $stmt->bind param('s', $category);

                //执行 SQL 语句
                $stmt->execute();

                //获取已插入记录对应的 catid
                $insertID = $stmt->insert_id;

                //关闭当前查询
                $stmt->close();

                //输出语句执行结果
                printf('Category %s (ID: %s) has been added.', $category,
                $insertID);
            }
        } else {
            //准备删除指定记录
            if($stmt = $dbConn->prepare('DELETE FROM bookcategory WHERE
            category=?')){
                $stmt->bind_param('s', $category);
                $stmt->execute();
                printf('Category %s (ID: %s) has been deleted.',
                $category, $sCatID);
            }
        }
    }
?>
```

在上面这段脚本中，我们使用 SELECT 语句查询指定图书类目，并根据查询结果判断该类目是否存在。具体的方法是：将查询结果输出到指定的变量中，然后使用 isset()函数判断该变量是否已设置。如果已设置，则表示指定图书类目已存在，需要删除。否则，指定图书类目不存在，需要添加。

上面这段脚本中，我们使用到了 mysqli 对象和 mysqli_stmt 对象中的若干方法和属性，具体如下：

（1）$dbConn->prepare($query)

该方法可以用来为 SQL 查询做准备工作。在脚本中，我们使用 prepare 方法为查询、添加和删除记录做好了准备。在输入 SQL 语句时，可以使用“?”占位符埋入若干未指定的参数，然后使用 mysqli_stmt 对象的 bind_param 方法指定这些参数。

（2）$stmt->bind_param($types, $param1,[$param2,…])

该方法可以指定 prepare 语句中使用“?”占位符进入的未指定的参数。

（3）$stmt->execute()

该方法可以执行准备好的 SQL 语句。需要注意的是，execute()方法的使用对象只能是 mysqli_stmt 对象。

（4）$stmt->bind_result($param1,[$param2,…])

该方法可以将查询结果中各字段的内容绑定到指定的变量或数组中。

（5）$stmt->fetch()

该方法用来获取已经绑定到指定的变量或数组中的查询结果。

（6）$stmt->insert_id

该属性可以用来获取上一条插入记录的 ID。这一属性对于有着自动增加 ID 字段的数据表来说，是十分有用的。

现在，我们再使用 MySQLi 扩展的面向过程的编程方法重新编写例 10.16-1 中的这段脚本。

【例 10.16-2】 使用 MySQLi 扩展面向过程的编程方法在指定数据表中添加和删除一条记录。

```
<?php
    $dbConn = @mysqli_connect('localhost', 'librarian', 'librarian9235',
    'librarymanager');
    if (mysqli_connect_error($dbConn)){
        die('Failed to connect to the database: '.mysqli_connect_
        error($dbConn));
    }

    $category = "Biography";

    If($stmt = mysqli_prepare($dbConn,'SELECT * FROM bookcategory WHERE
    category=?')){
        mysqli_stmt_bind_param($stmt, 's', $category);
        mysqli_stmt_execute($stmt);
        mysqli_stmt_bind_result($stmt, $sCatID, $sCat);
        mysqli_stmt_fetch($stmt);
        mysqli_stmt_close($stmt);
        if(!isset($sCatID)){
            if($stmt = mysqli_prepare($dbConn, 'INSERT INTO bookcategory
            SET category=?')){
                mysqli_stmt_bind_param($stmt, 's', $category);
                mysqli_stmt_execute($stmt);
                $insertID = mysqli_stmt_insert_id($stmt);
                mysqli_stmt_close($stmt);
                printf('Category %s (ID: %s) has been added.', $category,
                $insertID);
            }
        } else {
```

```
            if($stmt = mysqli_prepare($dbConn, 'DELETE FROM bookcategory
            WHERE category=?')){
                mysqli_stmt_bind_param($stmt, 's', $category);
                mysqli_stmt_execute($stmt);
                printf('Category %s (ID: %s) has been deleted.',
                $category, $sCatID);
            }
        }
    }
?>
```

在上面这段脚本里，我们使用了 MySQLi 扩展的面向过程的编程方法，实现了与例 10-16-1 中的脚本同样的功能。这段脚本中使用到了 MySQLi 扩展对象的如下几个方法和属性。

（1）mysqli_prepare($dbConn, $query)方法：相当于$dbConn->prepare($query)。

（2）mysqli_stmt_bind_param($stmt, $type, $param1[, $param2, …])方法：相当于$stmt->bind_param($type, $param1[, $param2, …])。

（3）mysqli_stmt_bind_result($stmt, $param1[, $param2, …])方法：相当于$stmt->bind_result($param1[, $param2, …])。

（4）mysqli_stmt_execute($stmt)方法：相当于$stmt->execute()。

（5）mysqli_stmt_fetch($stmt)方法：相当于$stmt->fetch()。

（6）mysqli_stmt_insert_id($stmt)属性：相当于$stmt->insert_id;。

下面我们来使用 PDO 扩展修改指定数据表中的指定记录。

【例 10.16-3】 使用 PDO 扩展修改指定数据表中的指定记录。

```
<?php
    try {
        $dbConn = new PDO('mysql:host=localhost;dbname=librarymanager',
        'librarian', 'librarian9235');

        $category = 'Linguistics';
        $changeTo = 'Language Studies';

        if($stmt = $dbConn->prepare('SELECT * FROM bookcategory WHERE
        category=:category')){
            $stmt->bindParam(':category', $category, PDO::PARAM_STR);
            $stmt->execute();
            $stmt->bindColumn(1, $sCatID);
            $stmt->fetch();
            if(!isset($sCatID)){
                printf('Category %s does not exist.', $category);
            } else {
                if($stmt = $dbConn->prepare('UPDATE bookcategory
                SET category=:changeto WHERE catid=:catid')){
                    $stmt->bindParam(':changeto', $changeTo, PDO::PARAM_STR);
                    $stmt->bindParam(':catid', $sCatID, PDO::PARAM_INT);
                    $stmt->execute();
                    printf('Category %s has been changed to %s', $category,
                    $changeTo);
                }
            }
        }
    } catch (PDOException $err) {
        echo $err -> getMessage();
    }
?>
```

执行这段脚本运行后，原来名为“Linguistics”的图书类目变成了“Language Studies”。在这段脚本中，我们首先查找指定的图书类目。通过判断该类目是否存在来决定下一步执行的脚本：若存在，则执行修改；若不存在，则输出通知。

编程中，我们使用了 PDO 扩展的 PDO 对象和 PDOStatement 对象中的若干方法如下所示。

（1）$stmt->bindParam($param, $variable, [$datatype])方法

该方法一次只能绑定一对参数与变量。除了使用之前提到的“?”占位符之外，我们还可以使用名称占位符，名称占位符的格式为“***:name***”。本例使用了后者。若需要使用“?”占位符，在指定$param 参数时，可输入代表查询语句中该参数所处位置的正整数。该方法相当于 MySQLi 扩展中的 mysqli_stmt_bind_param()方法。

（2）$stmt->bindColumn($column, $variable)方法

该方法一次只能绑定一对字段名与变量。在执行了准备好的 SQL 语句之后，我们可以将查询结果绑定到若干变量中。其中，$column 可以是字段名，也可以是该字段在数据表中所处位置的正整数。本例使用了后者。该方法相当于 MySQLi 扩展中的 mysqli_stmt_bind_result()方法。

（3）$stmt->execute()方法

该方法执行准备好的 SQL 语句。相当于 MySQLi 扩展中的 mysqli_stmt_execute()方法。

（4）$stmt->fetch()方法

该方法获取 SQL 语句执行后的结果集。相当于 MySQLi 扩展中的 mysqli_stmt_fetch()方法。

10.5　实战练习：记账工具（下）

在前两章中，我们为记账工具编写了一个名为 BookKeeping 的 PHP 类，并且搭建好了工具的用户界面。在本节里，将为该记账工具规划数据库、制定批量导入功能中使用的模板文件、并为这些页面添加相应的功能。现在就让我们一步一步来实现奇迹吧。

10.5.1　规划数据库

通过之前规划的用户界面，我们的数据库主要包括以下几张表格。图 10-17 展示了数据库中包含的数据表以及数据表之间的关系。

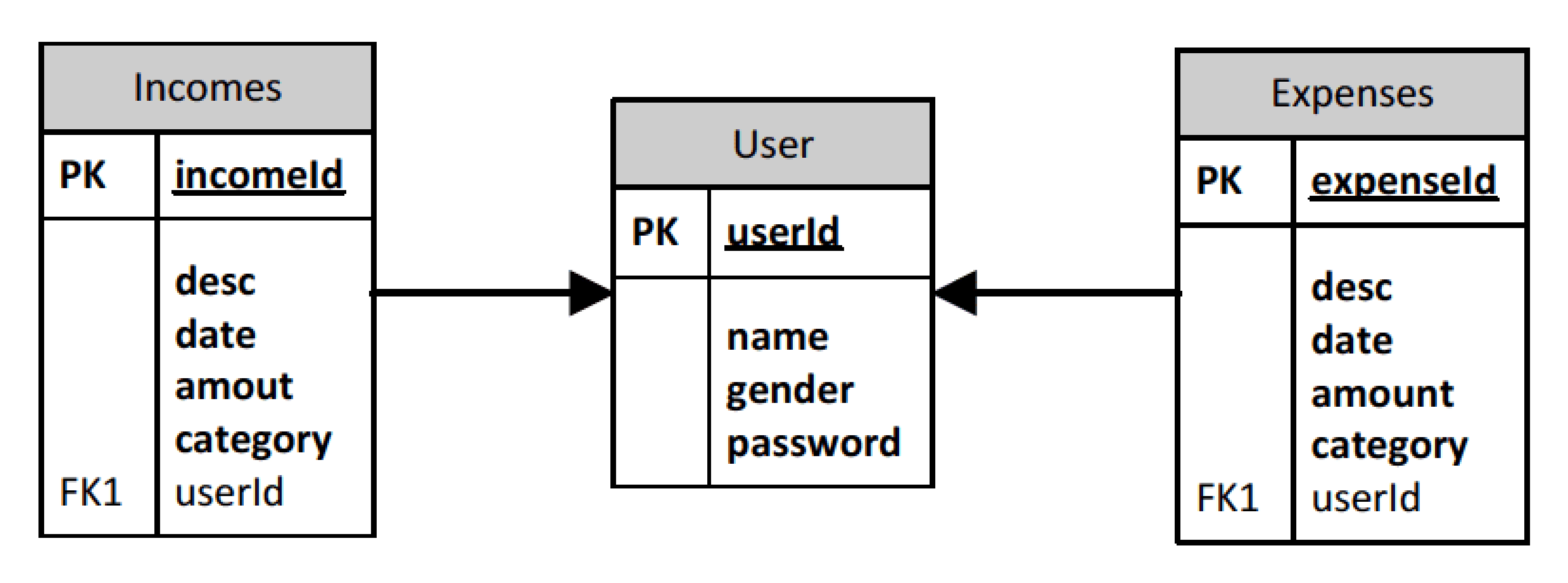


图 10-17　记账工具数据库表及表间关系

由于规划的功能都比较基础，所以数据表不多，只有三张，它们分别是 User 表、Incomes 表和 Expenses 表。其中，Incomes 和 Expenses 两张表中都有一个名为 UserId 的外键，表明了这三张表之间的关系。

读者可以根据 10.3.1 小节学习到的知识在 MySQL 服务器上创建一个数据库 bookkeeping，然后在数据库中创建这三张表格。

10.5.2　批量导入模板

由于我们批量导入的时候需要让用户在模板中填写若干条记录。然后，通过上传该模板文件，批量导入若干条记录。在本章学习了如何使用文本文件和 XML 文件存取数据。考虑到我们规划的功能比较简单，所以在这里使用文本文件来规划这个模板。

在该文本文件中，用户只需要输入类似下面的内容就可以完成一条记录：

```
...
{E|DE}手机 1 部
{E|AM}2588.63
{E|CA}用
...
{I|DE}7 月工资
{I|AM}8566.63
{I|CA}工资
...
{E|DE}午饭
{E|AM}16.50
{E|CA}食
```

上面这个模板文件定义了三条账务记录。其中，第一条和第三条为支出记录，第二条为收入记录，两条记录之间用“...”分隔。我们将每一条记录的三个字段分别写入三行中，并以“{*|**}”来标识每一行数据代表的意义。比如，“{E|DE}”标识了一条支出记录的说明，而“{I|AM}”标识了一条收入记录涉及的金额。

大家可以将上面的账务记录复制至一个文本文件中。需要注意的是，如果读者使用的是 Windows 操作系统，在保存时，务必用 UTF-8 的编码格式。操作方法如图 10-18 所示。

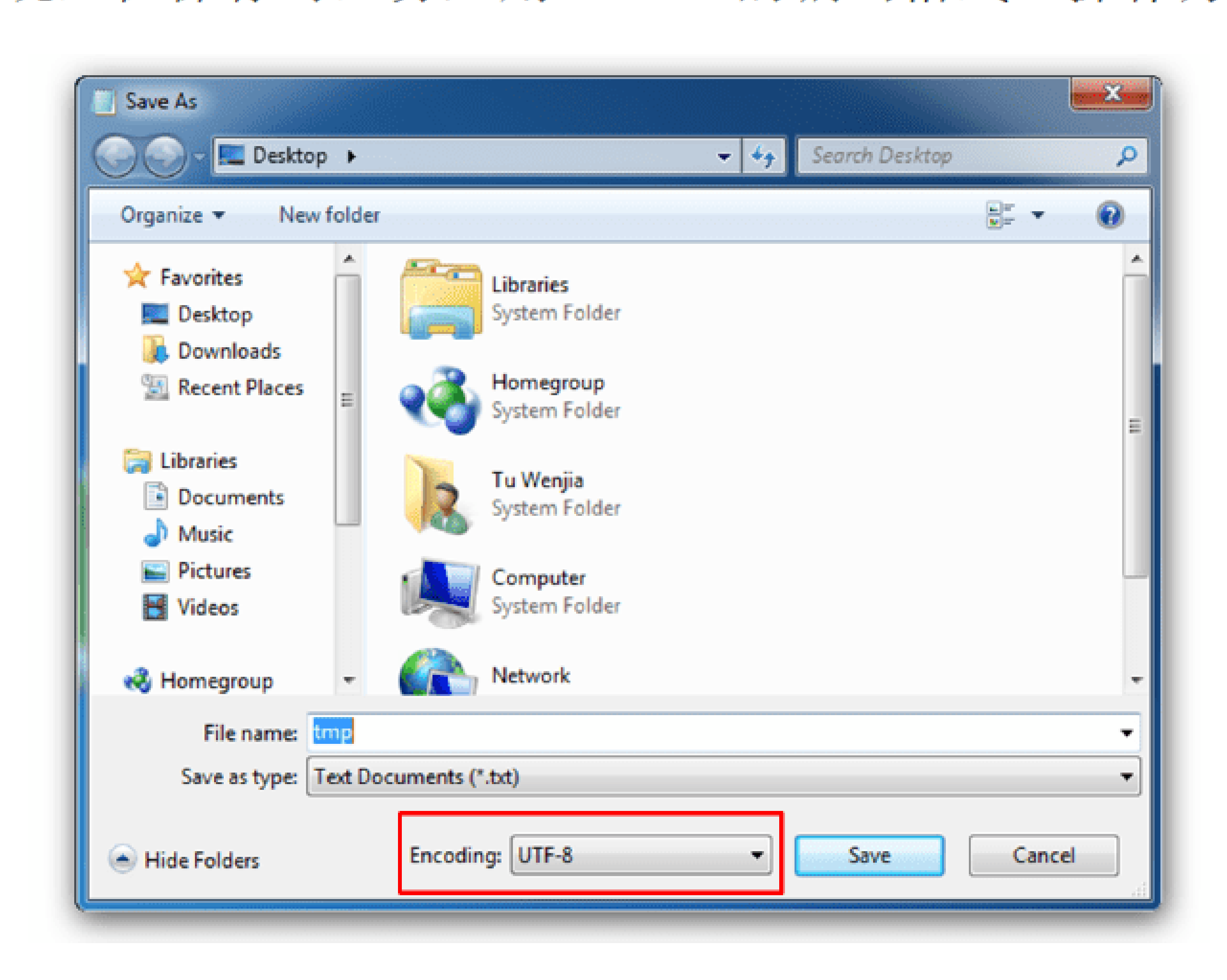


图 10-18　在 Windows 操作系统中使用 UTF-8 编码保存文本文件

10.5.3　为页面添加功能前的准备工作

在为各页面添加功能之前，再来考量一下第 8 章编写的那个 bookKeeping 类。在那个类中，定义了两个私有方法，分别为 AddIncomeToDatabase()和 AddExpenseToDatabase()。现在我们就来编写这两个私有方法。

```
……
    private function addExpenseToDatabase($desc,$amount,$category,$userId){
            //获取数据库连接
            $dbconn = $this -> getDBconn();
            //设置数据库字符集，用于处理中文字符
            $dbconn->set_charset('utf8');
            //使用 sprintf()函数拼接 SQL 查询语句
            $q = sprintf('SELECT expenseId FROM expenses
            WHERE eDesc="%s" AND eAmount =%s AND eCategory = "%s"',
            $desc,$amount,$category);
            //若上述拼接的 SQL 查询语句有结果，则取出结果；否则，中止脚本
            if($stmt = $dbconn->prepare($q)){
                $stmt->execute();
                $stmt->bind_result($expenseId);
                $stmt->fetch();
                $stmt->close();
            } else {
                die('NOK');
            }
            //若未查到满足条件的 expenseId，开始向数据库中插入记录
            if(!isset($expenseId)){
                $q = sprintf('INSERT INTO expenses (eDesc,eDate,eAmount,
                eCategory,userId)
                VALUES ("%s",CURRENT DATE,%f,"%s",%d)',$desc,$amount,
                $category,$userId);

                if($stmt = $dbconn->prepare($q)){
                    $stmt->execute();
                    $stmt->close();
                } else {
                    die('NOK');
                }
                //插入记录完成后，返回'Added'
                return 'Added';
            } else {
                //若查找到满足条件的 expenseId，则跳过插入记录，直接返回'Denied'
                return 'Denied';
            }
            //关闭数据库连接
            $dbconn->kill();
            $dbconn->close();
        }
……
        private function addIncomeToDatabase($desc,$amount,$category,
        $userId){
            //获取数据库连接
            $dbconn = $this -> getDBconn();
            //设置数据库字符集
            $dbconn->set charset('utf8');
            //使用 sprintf()函数拼接 SQL 查询语句
```

```
            $q = sprintf('SELECT incomeId FROM incomes
            WHERE iDesc="%s" AND iAmount =%s AND iCategory = "%s"', $desc,
            $amount,$category);

            //若上述拼接的 SQL 查询语句有结果，则取出结果；否则，中止脚本
            if($stmt = $dbconn->prepare($q)){
                $stmt->execute();
                $stmt->bind_result($incomeId);
                $stmt->fetch();
                $stmt->close();
            } else {
                die('NOK');
            }

            //若未查到满足条件的 incomeId，开始向数据库中插入记录
            if(!isset($incomeId)){
                $q = sprintf('INSERT INTO incomes (iDesc,iDate,iAmount,
                iCategory,userId)
                VALUES ("%s",CURRENT DATE,%f,"%s",%d)',$desc,$amount,
                $category,$userId);

                if($stmt = $dbconn->prepare($q)){
                    $stmt->execute();
                    $stmt->close();
                } else {
                    die('NOK');
                }

                // 插入记录完成后，返回'Added'。
                return 'Added';
            } else {
                // 若查找到满足条件的 incomeId，则跳过插入记录，直接返回'Denied'
                return 'Denied';
            }

            $dbconn->kill();
            $dbconn->close();
        }
```

紧接着，我们还需要为 BookKeeping 类定义两个新的私有方法，分别用来删除指定的消费和收入记录。这两个私有方法定为 removeExpenseFromDatabase()和 removeIncomeFromDatabase()，代码如下：

```
private function removeExpenseFromDatabase($expenseID) {
    $dbconn = $this -> getDBconn();
    $dbconn -> set charset('utf8');

    $q = sprintf('DELETE FROM expenses WHERE expenseID = %s', $expenseID);
    try {
        $stmt = $dbconn->prepare($q);
        $stmt->execute();
    } catch (Exception $e) {
        $msg = "Failed";
        die();
    }

    $dbconn->kill();
    $dbconn->close();

    $msg = "Removed";
```

```
}

private function removeIncomeFromDatabase($incomeID) {
    $dbconn = $this -> getDBconn();
    $dbconn -> set charset('utf8');

    $q = sprintf('DELETE FROM incomes WHERE incomeID = %s', $incomeID);
    try {
        $stmt = $dbconn->prepare($q);
        $stmt->execute();
    } catch (Exception $e) {
        $msg = "Failed";
        die();
    }

    $dbconn->kill();
    $dbconn->close();

    $msg = "Removed";
}
```

除了上述四个私有方法之外，还需要定义两个用于修改现有消费和收入记录的私有方法。这两个私有方法定名为 modifyExpenseInDatabase()和 modifyIncomeInDatabase()。它们的代码如下：

```
private  function  modifyExpenseInDatabase  ($expenseID,$desc,$amount,
$category){
    $dbconn = $this -> getDBconn();
    $dbconn -> set_charset('utf8');

    $q = sprintf('UPDATE expenses SET eDesc = "%s",
                 eAmount = %f,
                 $eCategory = "%s" WHERE expenseID = %s', $desc, $amount,
                 $category, $expenseID);

    try {
        $stmt = $dbconn->prepare($q);
        $stmt->execute();
    } catch (Exception $e) {
        $msg = "Failed";
        die();
    }

    $dbconn -> kill();
    $dbconn -> close();

    $msg = "Modified";

    return $msg;
}

private function modifyIncomeInDatabase ($incomeID,$desc,$amount,$category){
    $dbconn = $this -> getDBconn();
    $dbconn -> set_charset('utf8');

    $q = sprintf('UPDATE incomes SET iDesc = "%s",
                 iAmount = %f,
                 $iCategory = "%s" WHERE incomeID = %s', $desc, $amount,
                 $category, $incomeID);

    try {
```

```
            $stmt = $dbconn->prepare($q);
            $stmt->execute();
        } catch (Exception $e) {
            $msg = "Failed";
            die();
        }

        $dbconn -> kill();
        $dbconn -> close();

        $msg = "Modified";

        return $msg;
}
```

为了方便在页面上引用上述四个私有方法，我们创建四个新方法，用来修改和删除用户指定的数据，并在这四个新方法中引用上述四个私有方法。这四个新方法如下所示。

（1）ModifyExpenseEntry/ModifyIncomeEntry

在这两个新方法中，我们对用户提交的某记录修改后的数据进行检查，并将符合要求的数据存入数据库的指定记录。脚本如下：

```
function ModifyExpenseEntry($expenseId,$desc,$amount,$category){
    if(mb_strlen($desc,'UTF-8') <= 70 &&
       preg_match('/^\d+(\.\d+)?$/',$amount) &&
       strlen($userId) > 0) {
         $note = $this -> modifyExpenseInDatabase($expenseId, $desc,
         $amount, $category);
         if($note == "modified"){
             return '修改消费记录【消"'.sprintf('%05s',$expenseId).'"】成功。';
         } else {
             return '修改消费记录【消"'.sprintf('%05s',$expenseId).'"】失败。';
         }
       }
}

function ModifyIncomeEntry($incomeId,$desc,$amount,$category){
    if(mb_strlen($desc,'UTF-8') <= 70 &&
       preg_match('/^\d+(\.\d+)?$/',$amount) &&
       strlen($userId) > 0) {
         $note = $this -> modifyIncomeInDatabase($incomeId, $desc,
         $amount, $category);
         if($note == "modified"){
             return '修改收入记录【EXP"'.sprintf('%05s',$expenseId).'"】成功。';
         } else {
             return '修改收入记录【EXP"'.sprintf('%05s',$expenseId).'"】失败。';
         }
       }
}
```

（2）DeleteExpenseEntry/DeleteIncomeEntry

在这两个新方法中，我们将查询用户欲删除的记录是否存在。只有当用户指定的记录存在时，才会引用 removeExpenseFromDatabase 和 removeIncomeFromDatabase 两个私有方法删除指定的记录。脚本如下：

```
function DeleteExpenseEntry($expenseId){
    $dbconn = $this -> getDBconn();
    $dbconn->set_charset('utf8');
```

```
    $q = sprinf("SELECT * FROM expenses WHERE expenseId = %s", $expenseId);

    $msg = '';

    try{
        $stmt = $dbconn->prepare($q);
        $stmt->execute();
        $stmt->store result();
        if($stmt->num rows() == 1) {
            $note = $this->removeExpenseFromDatabase($expenseId);
            if($note == 'removed'){
                $msg .= '删除消费记录【EXP"'.sprintf('%05s',$expenseId).'"】
                成功。';
            } else {
                $msg .= '删除消费记录【EXP"'.sprintf('%05s',$expenseId).'"】
                失败。';
            }
        }
    } catch(Exception $e){
        $msg .= '系统内部错误，请尽快报告管理员。';
    }

    $dbconn -> kill();
    $dbconn -> close();

    return $msg;
}

function DeleteIncomeEntry($incomeId){
    $dbconn = $this -> getDBconn();
    $dbconn->set charset('utf8');

    $q = sprinf("SELECT * FROM incomes WHERE expenseId = %s", $expenseId);

    $msg = '';

    try{
        $stmt = $dbconn->prepare($q);
        $stmt->execute();
        $stmt->store result();
        if($stmt->num_rows() == 1) {
            $note = $this->removeIncomeFromDatabase($incomeId);
            if($note == 'removed'){
                $msg .= '删除收入记录【INC"'.sprintf('%05s',$incomeId).'"】
                成功。';
            } else {
                $msg .= '删除收入记录【INC"'.sprintf('%05s',$incomeId).'"】
                失败。';
            }
        }
    } catch(Exception $e){
        $msg .= '系统内部错误，请尽快报告管理员。';
    }

    $dbconn -> kill();
    $dbconn -> close();

    return $msg;
}
```

另外，为了获取用户指定记录的详细信息，还定义了一个 SearchForRecords 方法和 GetDetails 方法，前者需要用户指定日期范围和记录类型，后者则需要用户指定记录类型和记录编号。这两个方法的脚本如下所示。

（3）SearchForRecords 方法

```
function SearchForRecords($setYear, $setMonth, $recordType, $userId){
    //连接数据库
    $dbconn = $this -> getDBconn();
    $dbconn->set_charset('utf8');

    //计算时间点，基于此查询数据库中的记录
    $setDate = date('Y-m-d',mktime(0,0,0,intval($setMonth),1,intval
    ($setYear)));

    // 定义一个空数组
    $records = array();

    // 根据用户需要查找的记录类型在数据库中查找并输出记录
    if($recordType == 'EXP'){
        $q = sprintf('SELECT * FROM expenses WHERE eDate < "%s" AND userId
        = %s', $setDate,$userId);

        if($stmt = $dbconn -> prepare($q)){
            $stmt -> execute();
            $stmt->bind_result($expenseId, $eDesc, $eDate, $eAmount,
            $eCategory, $userId);
            $stmt->store_result();

            if ($stmt->num_rows > 0) {
                $i=0;

                while ($stmt->fetch()) {
                    $records[$i][0] = $expenseId;
                    $records[$i][1] = $eDesc;
                    $records[$i][2] = number_format($eAmount,2);
                    $records[$i][3] = $eCategory;
                    $records[$i][4] = $eDate;
                    $i++;
                }
            }
        }
    } else {
        $q = sprintf('SELECT * FROM incomes WHERE iDate < "%s" AND userId
        = %s', $setDate,$userId);

        if($stmt = $dbconn -> prepare($q)){
            $stmt -> execute();
            $stmt->bind_result($incomeId, $iDesc, $iDate, $iAmount,
            $iCategory, $userId);
            $stmt->store_result();

            if ($stmt->num_rows > 0) {
                $i=0;

                while ($stmt->fetch()) {
                    $records[$i][0] = $incomeId;
                    $records[$i][1] = $iDesc;
                    $records[$i][2] = number_format($iAmount,2);
                    $records[$i][3] = $iCategory;
```

```
                    $records[$i][4] = $iDate;
                    $i++;
                }
            }
        }
    }

    return $records;
}
```

（4）GetDetails 方法

```
function GetDetails($recordType,$recordId){
    $dbconn = $this -> getDBconn();
    $dbconn -> set_charset('utf8');

    $result = array();

    if($recordType == 'expense'){
        $q = sprintf('SELECT * FROM expenses WHERE expenseId = %s',
        $recordId);

        if($stmt = $dbconn->prepare($q)){
            $stmt->execute();
            $stmt->bind_result($expenseId,$eDesc,$eDate,$eAmount,
            $eCategory,$userId);

            while ($stmt -> fetch()) {
                $result[0] = $eDesc;
                $result[1] = number_format($eAmount, 2);
                $result[2] = $eCategory;
                $result[3] = $eDate;
            }
        }
    } else {
        $q = sprintf('SELECT * FROM incomes WHERE incomeId = %s', $recordId);
        if($stmt = $dbconn->prepare($q)){
            $stmt->execute();
            $stmt->bind_result($incomeId,$iDesc,$iDate,$iAmount,
            $iCategory,$userId);

            while($stmt->fetch()){
                $result[0] = $iDesc;
                $result[1] = number_format($iAmount, 2);
                $result[2] = $iCategory;
                $result[3] = $iDate;
            }
        }
    }

    return $result;
}
```

在上述与数据库操作相关的方法中，我们都使用了一个名为 getDBconn 的私有方法。接下来，定义这个名为 getDBconn()的私有方法用于获取与数据库之间的连接。另外，在批量上传的方法中，我们还使用了一个名为 readTemp()的方法用于处理用户上传的模板文件。这两个方法的脚本如下：

```
private function getDBconn(){
    $dbconn = @new mysqli('localhost','root','penpaper220','bookkeeping');
```

```
    if($dbconn->connect errno)
    {
        die('连接数据库失败：'.$dbconn->connect error);
    }

    return $dbconn;
}

//读取已上传文件的内容，并导入文件中的帐务记录
function readTemp($file,$userId){
    //判断用户是否已经成功上传了模板文件。若是，则将文件中的内容按行读取到$lines 数组中
    if(file_exists('upload\\'.$file)){
        $fh = fopen('upload\\'.$file, 'r');
        $lineCount = 0;
        while (!feof($fh)){
            if($lineCount == 0){
                $lines[$lineCount] = trim(substr(fgets($fh),3));
            }
            else {
                $lines[$lineCount] = trim(fgets($fh));
            }
            $lineCount++;
        }

        //设置一些参数
        $err = 0;           //记录模板文件中的错误数
        $eCount = 0;        //记录模板文件中可导入的消费记录数
        $iCount = 0;        //记录模板文件中可导入的收入记录数
        $msg = '';          //readTemp 方法返回消息

        //开始检查模板文件，记录文件中的错误数
        for($i=0;$i<count($lines);$i=$i+4){
            if($lines[$i] <> '...'){
                $err++;
            }
        }
        //若错误数为 0，则开始截取需要的数据，并引用之前定义的两个私有方法将数据添加到库中
        if($err == 0){

            /*模板文件中的各条记录间用“…”隔开，每个分隔行下三行为一条账务记录。所
            以下面循环的步长为$i = $i + 4 */

            for($i=1;$i<sizeof($lines);$i=$i+4){

                //检查每个分隔行下三行的起始六个字符是否分别为{E|DE}、{E|AM}和
                {E|CA}
                if(substr($lines[$i],0,6) == '{E|DE}' &&
                   substr($lines[$i+1],0,6) == '{E|AM}' &&
                   substr($lines[$i+2],0,6) == '{E|CA}'){

                    //若是，则获取各行第 6 个字符之后的内容存入对应的变量中
                    $desc = mb substr($lines[$i],6);
                    $amount = mb substr($lines[$i+1],6);
                    $category = mb substr($lines[$i+2],6);

                    //然后使用私有方法 addExpenseToDatabase 将记录添加到库中
                    $m = $this->addExpenseToDatabase($desc, $amount,
                    $category, $userId);
```

```
                    //若 addExpenseToDatabase 返回值为"Added", $eCount 值加 1
                    if($m == 'Added'){
                        $eCount++;
                    }

                }else if(substr($lines[$i],0,6) == '{I|DE}' &&
                        substr($lines[$i+1],0,6) == '{I|AM}' &&
                        substr($lines[$i+2],0,6) == '{I|CA}') {
                    $desc = mb_substr($lines[$i],6);
                    $amount = mb_substr($lines[$i+1],6);
                    $category = mb_substr($lines[$i+2],6);

                    $m = $this -> addIncomeToDatabase($desc, $amount,
                    $category, $userId);

                    if($m == 'Added'){
                        $iCount++;
                    }
                }
            }

            $msg .= '总共添加了'.$eCount.'条消费记录和'.$iCount.'条收入记录。';
        }else{
            $msg .= '模板中存在'.$err.'处错误。导入失败。';
        }
        return $msg;
    } else {
        return 'NOK';
    }
}
```

最后，我们向 bookKeeping 类中再添加一个新的方法，用来验证用户的身份，名字为 authenticate()。脚本如下：

```
function authenticate($username,$password){
        //获取数据库连接
        $dbconn = $this -> getDBconn();
        //查找数据库中是否存在用户名和密码匹配的记录
        $q = sprintf('SELECT userId FROM users where
        name = "%s" and password = "%s";',$username,$password);
        //若上述拼接的 SQL 语句可以运行，则运行该语句获取指定用户的用户 ID
        if($stmt = $dbconn->prepare($q)){
            $stmt->execute();
            $stmt->bind_result($id);
            $stmt->fetch();
            $stmt->close();
        }
        //若查找到指定用户的 ID，返回该 ID
        if(isset($id)){
            return $id;
        } else {
        //否则返回“NOK”。
            return 'NOK';
        }

        //关闭数据库连接
        $dbconn->kill();
        $dbconn->close();

    }
```

现在，我们就可以使用 bookKeeping 类的 authenticate 方法来验证用户身份了。还记得在 9.5.2 小节中在登录页面上添加了一段 PHP 脚本吗？使用这段脚本用来判断用户提交的信息，并据此将用户转向到指定的页面。这段脚本如下：

```
<?php
    session_start();
    if(isset($_REQUEST['login'])){
        $user = $_REQUEST['usr'];
        $pws = $_REQUEST['psw'];

        if(strlen($user) > 0) {
            $_SESSION['user'] = $user;
            header('location:./08.2_Book_Keeping_AddExpense.php');
        }
    }
?>
```

在这段脚本中，我们只是判断了用户是否输入了用户名，并没有通过查询数据库来判断数据库中是否存在该用户。现在来修改一下脚本：

```
if(isset($_REQUEST['login'])){
    $user = $_REQUEST['usr'];
    $pws = $_REQUEST['psw'];
    // 新建 bookKeeping 类对象
    $bk = new bookKeeping();
    // 使用$bk 对象的 authenticate()方法来获取指定用户的用户 ID
    $userId = $bk->authenticate($user,$pws);
    // 根据 authenticate()方法返回的内容决定用户的去向
    if($userId == 'NOK'){
        header('location:./08.1_Book_Keeping_Login.php');
    } else {
        $_SESSION['user']=$user;
        $_SESSION['userId']=$userId;
        header('location:./08.2_Book_Keeping_AddExpense.php');
    }
}
```

在上面这段脚本中，我们先定义了一个名为$bk 的 bookKeeping 类的对象，然后使用$bk 对象的 authenticate()方法检查了用户是否存在，并根据该方法的返回值判断用户是否可以进入其他页面。

现在，我们使用 phpMyAdmin 先在数据库的 user 表中添加一条记录，如图 10-19 所示。

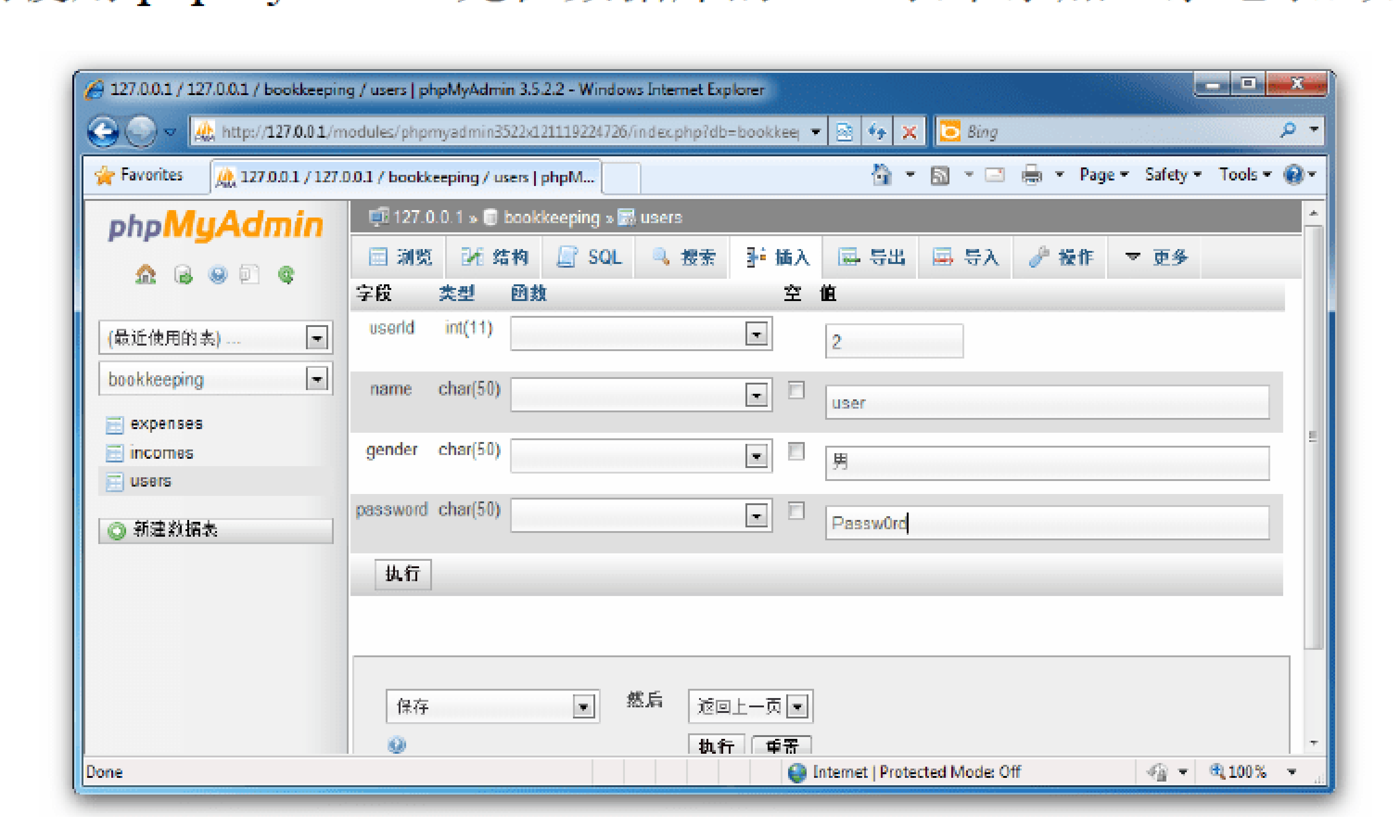


图 10-19　向数据库的 user 表中添加一条用户记录

接着，在浏览器中打开制作的记账工具的登录页面，在用户名和密码处分别输入“user”和“Passw0rd”。然后单击“立即登录”按钮，如图 10-20 所示。

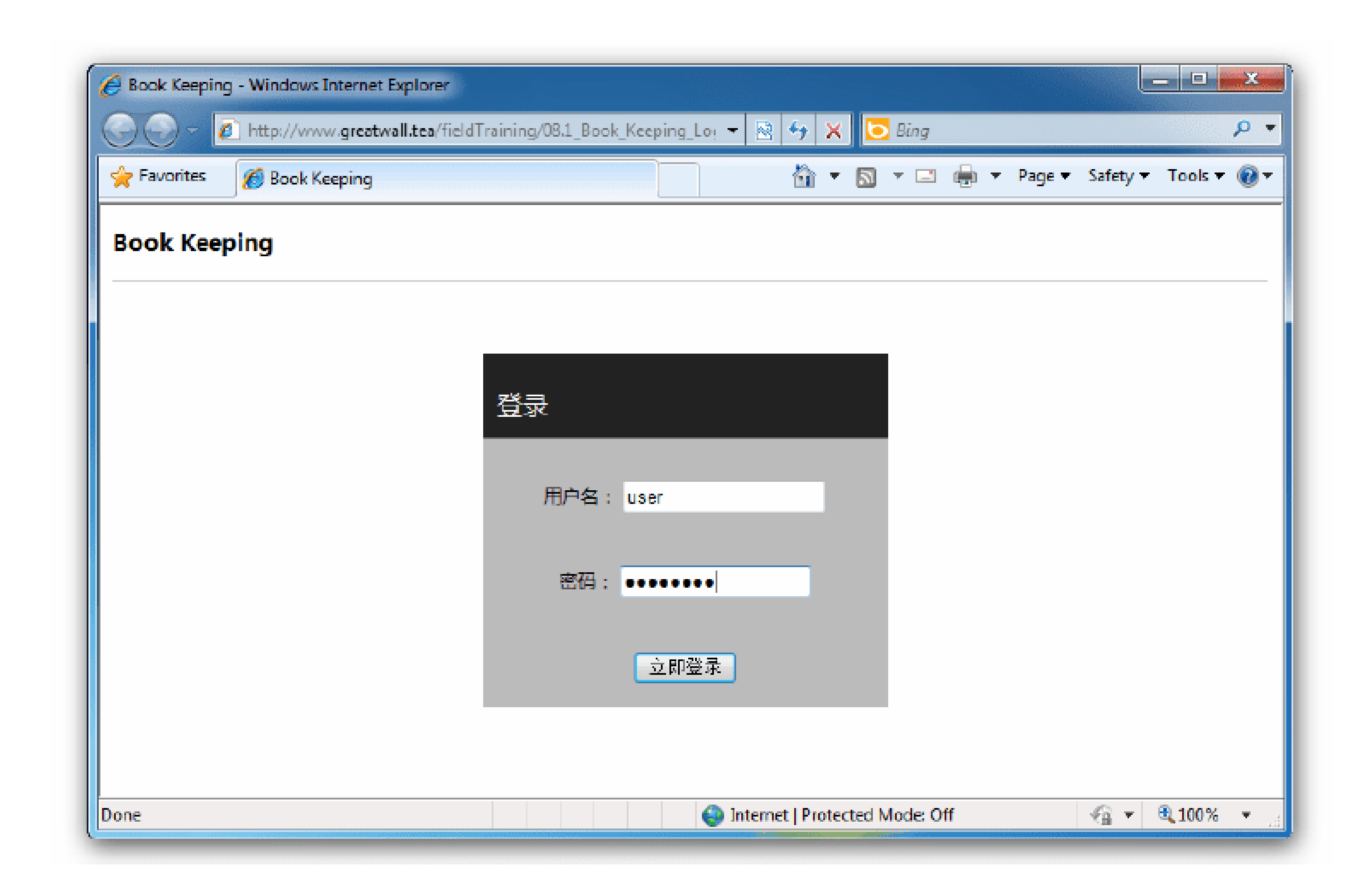


图 10-20　登录记账工具

到此，我们就完成了为页面添加功能前的准备工作。

10.5.4　为页面添加功能

1．为“添加消费记录”页面添加功能代码

在第 9 章里，在添加消费记录页面上添加了一张表单。表单中有一个文本域、一个下拉选框和一个文本框，分别用来输入记录描述、消费类型和消费金额。页面如图 10-21 所示。



图 10-21　添加消费记录

为了通过这个页面提交消费记录，我们可以在判断用户提交了表单之后，获取并保存用户填写的数据到相应的变量，然后将这些变量的值存入数据库的相应数据表中。

按照这个思路，在这个页面的</body>标签前添加如下 PHP 脚本：

```
<?php
    //判断用户是否提交了表单
    if(isset($_REQUEST['addExpense'])){
        //若用户提交了表单，则获取用户提交的数据
        $eDesc = $_REQUEST['desc'];
        $eCategory = $_REQUEST['category'];
        $eAmount = $_REQUEST['amount'];
        $userId = $_SESSION['userId'];

        $msg = '';

        //建立 bookKeeping 类的实例，用于添加消费记录
        $bk = new bookKeeping();

        $msg = $bk ->AddSingleExpense($eDesc, $eAmount, $eCategory, $userId);

        echo '<script language="javascript">document.getElementById ("msg").
        innerHTML="<p>'.$msg.'</p>"</script>';
    }
?>
```

在上面这段脚本中，我们首先判断用户是否提交了表单。若用户提交了表单，则获取用户提交的数据，并创建一个 bookKeeping 类的实例，并使用其 AddSingleExpense()方法，将用户提交的数据存入数据库中。

需要注意的是，AddSingleExpense()方法自带了数据验证的脚本，所以在这个页面上我们就没有验证用户输入的数据。

2．为“添加收入记录”页面添加功能代码

为添加收入记录的页面添加功能的思路与为添加消费记录的页面添加功能的思路是一样的。都是通过获取用户输入的数据并将其存入到数据库中。页面如图 10-22 所示。

图 10-22　添加收入记录

为了实现预定的功能，其脚本如下：

```
<?php
```

```
    //判断用户是否提交了表单
    if(isset($_REQUEST['addIncome'])){
        //若用户提交了表单，则获取用户提交的数据
        $iDesc = $_REQUEST['desc'];
        $iCategory = $_REQUEST['category'];
        $iAmount = $_REQUEST['amount'];
        $userId = $_SESSION['userId'];

        $msg = '';

        //建立 bookKeeping 类的实例，用于添加收入记录
        $bk = new bookKeeping();

        $msg = $bk -> AddSingleIncome($iDesc, $iAmount, $iCategory, $userId);

        echo '<script language="javascript">document.getElementById("msg").
        innerHTML="<p>'.$msg.'</p>"</script>';
    }
?>
```

3．为“批量上传导入记录”页面添加功能

批量上传可以说是这个记账工具中比较有特色的一个功能。使用批量上传，可以帮助用户使用规定的标记语言事先在文本文件中录入消费和收入记录，然后再一次性的导入到数据库中。这个页面如图 10-23 所示。



图 10-23　批量添加记录

在批量上传记录的页面上，我们在“</body>”标签前添加如下的 PHP 脚本：

```
<?php
    //检查用户是否点击了“开始上传”按钮。若是，则开始获取与上传文件相关的信息
    if(isset($_REQUEST['batchUpload'])){
        $temp_name = $_FILES['user_file']['tmp_name'];
        $name = $_FILES['user_file']['name'];
        $typeof = $_FILES['user_file']['type'];
        $userId = $_SESSION['userId'];
        $msg = '';

        //若临时文件不存在或文件格式不是文本文件，输出错误提示
        if($temp_name == '' ||
```

```
            !preg match('/text/',$typeof)){
             $msg = '上传失败或格式不正确。';
          } else {
             //否则，指定用于保存用户上传文件的文件夹路径
             $destination = dirname(__FILE__).'\upload\\'.$name;
             //移动临时文件至指定的该文件夹中
             move uploaded file($temp name, $destination);
             //新建名为$bk 的 bookKeeping 类对象
             $bk = new bookKeeping();
             //使用$bk 对象的 readTemp 方法处理用户上传文件中的数据
             $msg = $bk->readTemp($name, $userId);

          }
          //向 id 为"msg"的 div 标签中输出信息。
          echo '<script language="javascript">document.getElementById("msg").
          innerHTML=
          "<p>'.$msg.'</p>"</script>';
             }
?>
```

然后使用定义好的用户名和密码登录记账工具，单击页面左侧导航栏中的“批量导入记录”链接，进入“批量添加记录”页面。然后单击“浏览”按钮找到需要上传的文件。单击“开始上传”按钮，随后，在按钮的下方会出现处理结果，如图 10-24 所示。

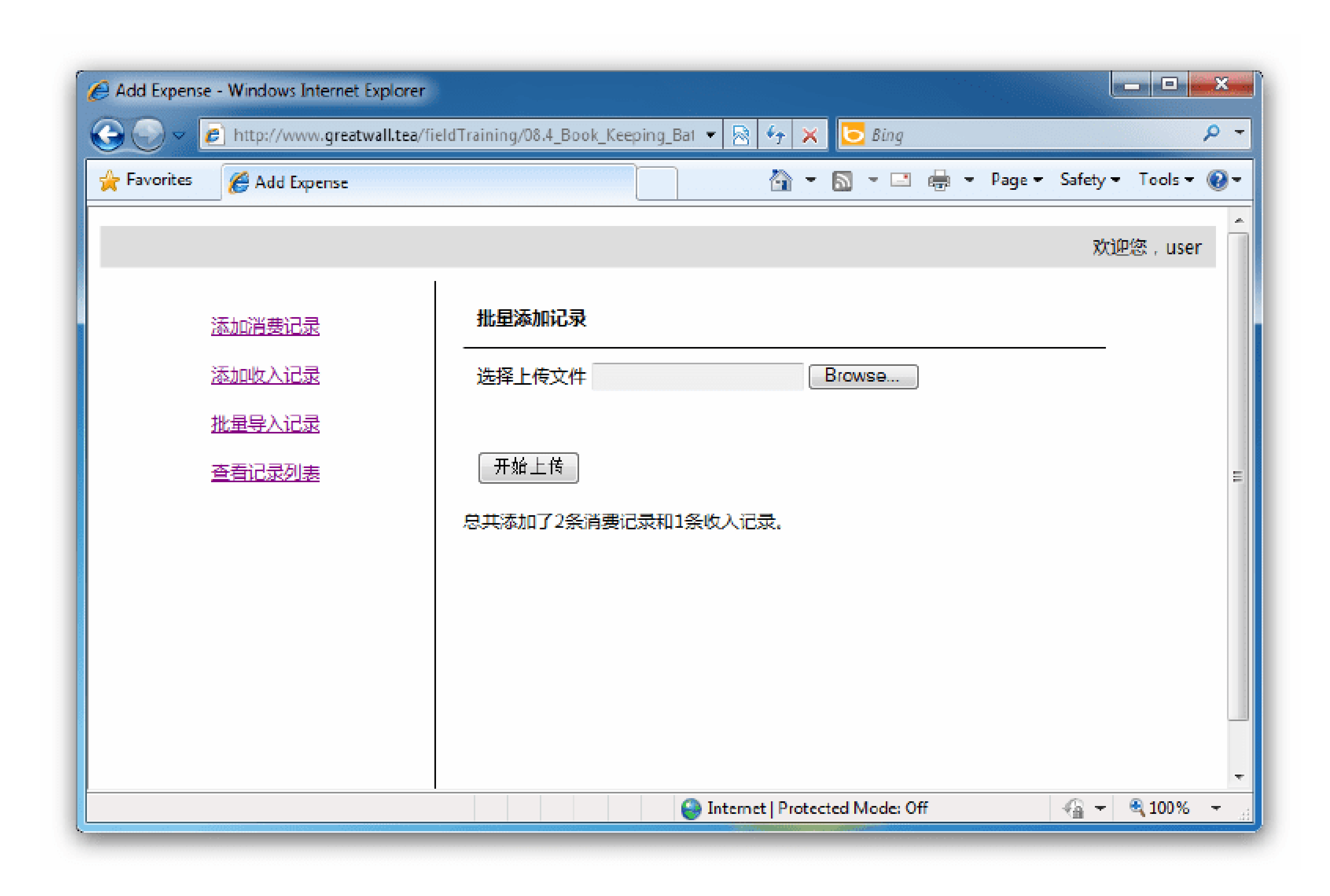


图 10-24　成功添加了 2 条消费记录和 1 条收入记录

随后可以通过 phpMyAdmin 查看数据库中是否出现了这三条属于名为“user”用户的账务记录。

至此，我们已经完成了记账工具的批量导入功能。剩余的几个页面，内容大同小异，读者可以尝试着自己完成。

4．为“查看记录列表”页面添加功能

在这个页面里，用户可以查询指定时间范围内的消费和收入记录，并对这些记录进行修改或删除其中的若干条记录。页面如图 10-25 所示。



图 10-25　查看记录列表

在查询到记录时，页面如图 10-26 所示。

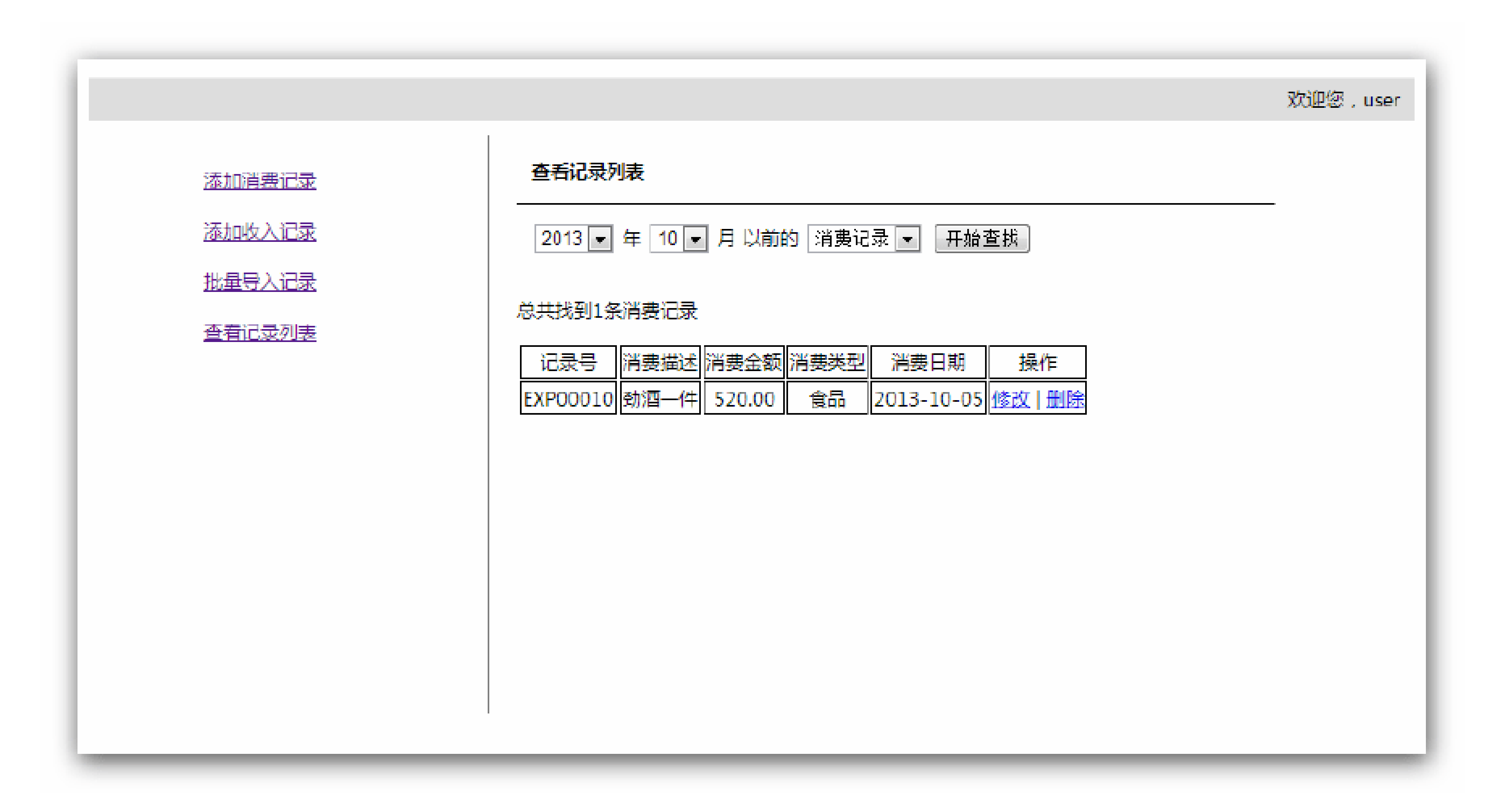


图 10-26　查询到记录时的“查询列表页面”

在查询到记录时，我们可以对查询到的记录做修改或删除的操作，为此还需要创建两个新页面分别用于修改用户指定的记录和删除用户指定的记录。

关于这两个页面的 HTML 部分，可以参考我们随书提供的源代码。这里重点来看看如何使用 PHP 实现查询、修改和删除。

（1）查询指定时间前的消费和收入记录

为了实现查询指定时间前的消费和收入记录，需要在查询页面中 ID 为 table 的 DIV 中添加如下代码：

```
<?php
    if(isset($_REQUEST['search'])){
        //获取用户提交的数据
        $setYear = $_REQUEST['year'];
        $setMonth = $_REQUEST['month'];
        $recordType = $_REQUEST['type'];
        $userId = $_SESSION['userId'];
```

```
//创建 bookKeeping 类实例，并使用该实例的 SearchForRecords 方法查找记录
$bk = new bookKeeping();
$records = $bk -> SearchForRecords($setYear,$setMonth,
$recordType, $userId);

//输出查询到的记录到表格中
if($recordType == 'EXP'){
    echo '<p>总共找到'.count($records).'条消费记录</p>';
    echo '<table><tr><td>记录号</td>
          <td>消费描述</td>
          <td>消费金额</td>
          <td>消费类型</td>
          <td>消费日期</td>
          <td>操作</td></tr>';

    if(count($records) > 0) {
        foreach ($records as $key => $value) {
            echo '<tr><td>EXP'.sprintf('%05s',$value[0]).'</td>
                 <td>'.$value[1].'</td>
                 <td>'.$value[2].'</td>
                 <td>'.$value[3].'</td>
                 <td>'.$value[4].'</td>
                 <td>
                   <a href="08.6 Book Keeping Modify.php
                   ?t=expense&id='.$value[0].'">修改</a> |
                   <a href="08.7_Book_Keeping_Remove.php
                   ?t=expense&id='.$value[0].'">删除</a>
                 </td></tr>';
        }
    } else {
        echo '<tr><td colspan="6">未找到相应消费记录。</td></tr>';
    }

    echo '</table>';
} else {
    echo '<p>总共找到'.count($records).'条收入记录</p>';
    echo '<table><tr><td>记录号</td>
          <td>收入描述</td>
          <td>收入金额</td>
          <td>收入类型</td>
          <td>收入日期</td>
          <td>操作</td></tr>';

    if(count($records) > 0) {
        foreach ($records as $key => $value) {
            echo '<tr><td>INC'.sprintf('%05s',$value[0]).'</td>
                 <td>'.$value[1].'</td>
                 <td>'.$value[2].'</td>
                 <td>'.$value[3].'</td>
                 <td>'.$value[4].'</td>
                 <td>
                   <a href="08.6_Book_Keeping_Modify.php
                   ?t=income&id='.$value[0].'">修改</a> |
                   <a href="08.7 Book Keeping Remove.php
                   ?t=income&id='.$value[0].'">删除</a>
                 </td></tr>';
        }
```

```
            } else {
                echo '<tr><td colspan="6">未找到相应收入记录。</td></tr>';
            }

            echo '</table>';
        }
    }
?>
```

在上面这段代码中，我们主要做了三件事。第一件是判断用户是否提交了表单，并获取用户提交的数据。第二件是创建 bookKeeping 类实例，并使用该实例的 SearchForRecords 方法来查找用户指定时间之前的指定类型的记录，并将查询到的内容存入数组$records 中。第三件是根据用户指定的记录类型，遍历数组$records，并生成表格。

在这三件事中，第一件事情与在其他页面上的判断类似，第二件事情，应该在为页面添加功能前的准备工作中已经完成了。因此，第三件事情就是此脚本的重点。由于此处只是巩固我们之前学习到的内容，所以把 HTML 代码和 PHP 脚本糅合在了一起。在实际开发的最佳实践中，我们应该把呈现层（即 HTML 代码）和控制层（即 PHP 脚本）分离开来，以便将来的脚本维护工作。

在根据用户指定的时间范围和记录类型查找到相关记录后，我们可以通过每条记录的操作栏中提供的“修改”和“删除”链接来修改和删除该记录。

（2）修改指定的消费和收入记录

为了实现修改指定消费和收入记录，我们需要使用 bookKeeping 类的 ModifyExpenseEntry 和 ModifyIncomeEntry 两个方法。这两个方法都要求用户提供四个参数：前者要求的参数有消费记录编号（expenseId）、新消费描述（eDesc）、新消费金额（eAmount）及新消费类型（eCategory）；而后者要求的参数为收入记录编号（incomeId）、新收入描述（iDesc）、新收入金额（iAmount）及新收入类型（iCategory）。

另外，在查询页面中，用户既可以查询消费记录，也可以查询收入记录，在修改记录的时候，我们也需要通知修改页面需要修改记录到底是消费记录还是收入记录。因此，需要使用查询页面中的“修改”链接的 URL 向修改记录的页面传递“记录类型”和“记录编号”两个参数，然后在修改记录的页面中根据传递过来的“记录类型”和“记录编号”查找相应记录。

综上所述，我们需要在修改记录的页面上完成三件事：第一件事是获取用户通过 URL 传递过来的“记录类型”和“记录编号”；第二件事情是根据这两个参数查询记录的详情，并将详情各项填充到用户修改记录时使用的表单中；第三件事情是处理用户更新的数据。为此，我们需要在修改记录的页面的不同位置插入 PHP 脚本。具体如下：

为了完成第一件事，我们需要在页面的头部插入如下 PHP 脚本：

```
if(isset($_SESSION['user'])) {
    $recordType = $_REQUEST['t'];
    $recordId = $_REQUEST['id'];

    $title = ($recordType == "expense") ? '消费记录' : '收入记录';

    $bk = new bookKeeping();

    $details = $bk -> GetDetails($recordType, $recordId);
```

在这段脚本中，我们使用$_REQUEST 数组获取了用户通过 URL 传递过来的数据。然后创建了一个 bookKeeping 类实例，并使用该实例的 GetDetails 方法获取了用户指定记录的详情。

为了完成第二件事，我们将获取到的详情填充到了用户在修改记录时需要用到的表单中。这一部分糅合 HTML 代码和 PHP 脚本。看上去虽然复杂，但其他原理很简单，那就是使用 echo 方法将获取到的详情各项“输出”到对应的表单元素的“Value”属性中。具体脚本可以参考我们随书提供的源代码。

为了完成第三件事，我们将在修改记录的页面的</body>标签前添加如下 PHP 脚本：

```
<?php
    //判断用户是否提交了表单
    if(isset($_REQUEST['modRecord'])){
        //若用户提交了表单，则获取用户通过表单和 URL 地址传递的所有参数
        $recordType = $_REQUEST['t'];
        $recordId = $_REQUEST['id'];
        $modDesc = $_REQUEST['desc'];
        $modCategory = $_REQUEST['category'];
        $modAmount = $_REQUEST['amount'];

        //创建一个 bookKeeping 类的实例，并根据变量$recordType 的值执行不同的修改方法
        $bk = new bookKeeping();
        if($recordType == 'expense'){
            $msg = $bk->ModifyExpenseEntry($recordId, $modDesc,
                        $modAmount, $modCategory);
        } else {
            $msg = $bk->ModifyIncomeEntry($recordId, $modDesc,
                        $modAmount, $modCategory);
        }

        //将修改结果返回到页面中名为 msg 的 DIV 元素中
        echo '<script language="javascript">document.getElementById("msg").
        innerHTML="<p>'.$msg.'</p>"</script>';
    }
?>
```

这段脚本首先判断用户是否提供了表单。若用户提交了表单，则获取用户通过表单和 URL 地址传递的所有参数。然后创建一个 bookKeeping 实例，并使用该实例的 ModifyExpenseEntry 方法或 ModifyIncomeEntry 方法将获取到的参数更新到数据库中的指定记录里。

（3）删除指定的消费和收入记录

为了实现删除指定的消费和收入记录，我们需要使用 bookKeeping 类的 DeleteExpenseEntry 和 DeleteIncomeEntry 方法。这两个方法均只要求提供一个参数，即用户指定记录的记录编号。这也就是说，一旦执行这两个方法删除指定的记录且记录存在时，记录就会直接被删除。

为了提供给用户一个后悔的机会，我们需要在删除记录的页面上添加一个“确认”机制，在用户单击记录列表中的“删除”按钮来到删除页面时，首先会看到一个确认框（如图 10-27 所示），用户可以在这个确认框中看到其指定的记录的详细内容。如果确认需要删除这个记录，则可以单击确认框下方的“删除”链接删除该记录。反之，则可以单击确认框下方的“取消”链接返回查询页面。



图 10-27　删除记录前的确认机制

为了实现这个功能，我们需要在页面的头部添加如下 PHP 脚本：

```
//判断用户是否为已登录用户
if(isset($_SESSION['userId'])){
    //若为已登录用户，则获取用户通过 URL 传递到本页面的参数
    $recordType = $_REQUEST['t'];
    $recordId = $_REQUEST['id'];

    $title = ($recordType == 'expense') ? "消费记录" : "收入记录";

    //新建一个 bookKeeping 实例
    $bk = new bookKeeping();

    //判断用户是否确认删除指定的记录
    if (isset($_REQUEST['confirm'])){
        //根据用户提交的记录类型，来执行 bookKeeping 实例的方法
        if ($recordType == 'expense'){
            $msg = $bk->DeleteExpenseEntry($recordId);
        } else {
            $msg = $bk->DeleteIncomeEntry($recordId);
        }
    } else {
        $record = $bk->GetDetails($recordType, $recordId);

        //判断指定记录是否存在。由于 GetDetails 方法返回值为数组，这里只需要判断变量
        $record 元素个数是否为 0
        if(count($record) > 0){
            // 若数组元素个数不为 0，则开始输出获取到的信息
            if($recordType == 'expense'){
                $msg = '<p>确定要删除如下消费记录吗？'.'</p>';
                $msg .= '<ul><li>[记录号] EXP'.sprintf('%05s',$recordId).
                '</li>';
                $msg .= '<li>[消费描述] '.$record[0].'</li>';
                $msg .= '<li>[消费金额] '.$record[1].'</li>';
                $msg .= '<li>[消费类型] '.$record[2].'</li>';
                $msg .= '<li>[消费日期] '.$record[3].'</li></ul>';

            } else {
                $msg = '<p>确定要删除如下收入记录吗？</p>';
```

```
                $msg .= '<ul><li>[记录号] EXP'.sprintf('%05s',$recordId).
                '/li>';
                $msg .= '<li>[收入描述] '.$record[0].'</li>';
                $msg .= '<li>[收入金额] '.$record[1].'</li>';
                $msg .= '<li>[收入类型] '.$record[2].'</li>';
                $msg .= '<li>[收入日期] '.$record[3].'</li></ul>';

            }
        }
    }
```

在上面这段脚本中，我们获取了用户通过 URL 传递的参数，然后创建一个 bookKeeping 类的实例并使用其 GetDetails 方法来获取用户指定记录的详情，然后将详情存入变量$msg 中。若用户确认要删除显示的记录，则将删除记录的方法返回的消息存入变量$msg 中。

接着我们在页面中 ID 为“confirm-box”的 DIV 中，插入如下脚本：

```
<?php echo $msg; ?>
<p id="right-align">
    <?php if (isset($_REQUEST['confirm'])) { ?>
    <a href="08.2_Book_Keeping_AddExpense.php">返回首页</a>
    <?php } else { ?>
    <a href="<?php echo '?t='.$recordType.'&id='.$recordId.'&confirm'; ?>">
    删除</a>
    <a href="08.5_Book_Keeping_Query.php">取消</a>
    <?php } ?>
</p>
```

在这段脚本中，我们输出了变量$msg 的值，并根据用户是否已确认删除显示的记录来决定确认框下方显示的链接。若用户处于确认阶段，则确认框下方的链接为“删除”和“取消”。单击“删除”可删除显示的记录，单击取消可以回退到记录查询页面。若用户已经确认删除显示的记录，则确认框正文的链接为“返回首页”。单击该链接可以回到“添加消费记录”页面。

10.6　习　　题

（1）使用文本文件做为存储介质，编写一个日记本应用。

（2）使用 XML 文件做为存储介质，编写一个图库应用。

（3）使用 MySQL 数据库为数据存储介质，编写一个用户信息管理系统。

第 4 篇　面向对象编程

第 11 章　PHP 与操作系统

PHP 为我们提供了大多数编程语言有的文件操控功能。使用 PHP，可以很方便获取操作系统及存储在操作系统中的文件信息。具体说来，我们可以使用 PHP 在操作系统中创建、复制、删除、查找和移动文件，也可以使用 PHP 运行操作系统中的任何程序。同样，我们还可以在相互连接的两台电脑间通过 PHP 或电子邮件传送消息和文件。

在本章中就来看看如何使用 PHP 来实现这一切。

11.1　管 理 文 件

信息都是以文件的形式存储在硬盘上的。但是为了便于管理，我们把这些文件分门别类地放入不同的文件夹内。由文件和文件夹组成的系统称为文件系统。通常来说，一个文件系统的结构是分层的。在这个分层结构的顶端只有一个文件夹，称为根文件夹。在 Windows 系统中根文件夹为“C:\”，而在 Linux 系统中根文件夹为“/”。所有的文件和文件夹都存放在根文件夹中。文件夹中可以包含文件和子文件夹。理论上来说，一个文件夹可以包含的文件夹层级是无限的。

使用文件夹来管理文件是一件很逻辑的事情。然而，对于实现文件夹的方式来说，其本身其实也是一种文件。它是一份关于存储在该文件夹中的文件和子文件夹的列表。通过该列表，操作系统可以很方便地找到相应的文件和子文件夹。

正如在上一章里学到的，PHP 为我们提供了读取、修改和删除文件的方法。除此之外，还可以使用 PHP 对文件和文件夹进行检查、删除、复制和重命名等操作。在本章里，我们会涉及到几乎所有与文件管理相关的方法，但是 PHP 为我们提供的方法却不仅限于此。如果想了解更多，可以在 PHP 官方网站上的手册中查找。如果找不到想要的功能，而该功能可以通过系统中的其他某些应用程序或命令来实现，就可以通过调用操作系统命令来完成相应的任务。

11.1.1　获取文件信息

我们需要了解某些文件的相关信息。比如，对于一张图片来说，可能需要知道这张图片有多大、尺寸是多少、什么时间以什么格式存放在系统中。如果这张图片是一张数码相机拍摄的像片，还需要了解这张照片是用什么设备拍摄的、光圈是多少等更加详细的信息。

我们可以使用 file_exists()方法来判断某个文件是否存在，该方法返回一个逻辑值。比如：

```
$result = file exists('stuff.txt');
if (!$result)
{
echo "file not found!";
}
```

当我们得知文件存在时，就可以查看其中的信息了。表 11-1 列出了常用的与文件管理相关的函数和方法。

表 11-1　常用的文件管理函数

模　式	说　明	输　出
is_file("stuff.txt")	判断指定的对象是否为一个文件，而不是文件夹或其他特殊格式的文件	TRUE 或者 FALSE
is_dir("stuff.txt")	判断指定的对象是否为一个文件夹	TRUE 或者 FALSE
is_executable("stuff.txt")	判断指定的对象是否为一个可执行文件	TRUE 或者 FALSE
is_writable("stuff.txt")	判断指定的对象是否可写	TRUE 或者 FALSE
is_readable("stuff.txt")	判断指定的对象是否可读	TRUE 或者 FALSE
fileatime("stuff.txt")	返回最近一次访问指定文件的时间	UNIX 时间戳或 FALSE
filectime("stuff.txt")	返回创建该文件的时间	UNIX 时间戳或 FALSE
filemtime("stuff.txt")	返回最近一次修改指定文件的时间	UNIX 时间戳或 FALSE
filegroup("stuff.txt")	返回指定文件所属组的 ID	代表文件组 ID 的整数或 FALSE
fileowner("stuff.txt")	返回指定文件的所有者 ID	代表文件所有者 ID 的整数或 FALSE
filesize("stuff.txt")	返回指定文件的大小，单位为字节	代表文件大小的整数或 FALSE
filetype("stuff.txt)"	返回指定文件的类型	代表文件类型的字符串（取舍可能为 file、dir、file、link 和 char）或 FALSE
basename("/dir/stuff.txt")	返回指定路径中的文件名	stuff.txt
dirname("/dir/stuff.txt")	返回指定路径中的路径	/dir

除了上面这些函数之外，我们还可以使用 pathinfo()来获取一个描述文件在系统中的位置的数组。比如：

```
$pinfo = pathinfo("/dir/stuff.txt");

echo $pinfo[dirname];            //输出“/dir”
echo $pinfo[basename];           //输出“stuff.txt”
echo $pinfo[extension];          //输出“txt”
```

11.1.2　复制、重命名和删除文件

在上一章里，我们知道了如何创建一个文本文件、如何向文件中写入信息。在本小节来了解一下如何复制、重命名和删除文件。

需要注意的是，在使用 copy()函数进行文件复制操作时，我们只能复制已存在的文件。在复制完成后，会得到两个内容一样、文件名不一样的文件。通常在需要备份某个文件时使用复制操作。我们可以使用类似如下的脚本复制一个指定文件的内容到另一个文件中：

```
copy("fileold.txt","filenew.txt");
```

在 PHP 执行该语句后，fileold.txt 这个已存在于操作系统中的文件会被复制为

filenew.txt 这个文件。在执行过程中，PHP 会碰到两种情况：

- 若 filenew.txt 文件不存在，则 PHP 会主动创建 filenew.txt 文件并将 fileold.txt 文件的内容复制到 filenew.txt 中。
- 若 filenew.txt 文件已存在，则 PHP 会用 fileold.txt 文件的内容来覆盖 filenew.txt 文件的内容，也就是说，filenew.txt 文件原有的内容会消失。

若不想覆盖某个文件的内容，可以使用 file_exits()函数进行判断。比如：

```
if(!file_exists("filenew.txt")){
copy("fileold.txt","filenew.txt");
} else {
echo "file already exists!";
}
```

类似地，我们可以使用 rename()函数来重命名一个指定的文件，如：

```
rename("oldname.txt","newname.txt");
```

PHP 在执行该语句时，会进行如下的逻辑判断。

- 第一步　先判断 oldname.txt 文件是否存在。若存在，则进行第二步。若不存在，则发出如下警告：

```
Warning: rename(oldname.txt,newname.txt): The system cannot find the file
specified. (code: 2) in C:\greatwall\chapter11\Ex11-1.php on line 2
```

- 第二步　判断 newname.txt 文件是否存在。若存在，则发出如下警告；若不存在，则将 oldname.txt 的文件名替换为 newname.txt：

```
Warning: rename(oldname.txt,newname.txt): File exists in c:test.php on line 2.
```

类似地，如果想移除某个文件，可以使用 unlink()方法，如：

```
unlink("badfile.txt");
```

PHP 在执行该语句时，会判断指定文件是否存在。如果存在则删除该文件，若不存在，则报错。

现在来看一段脚本。在下面这段脚本中，我们使用 fopen()函数、copy()函数、rename()函数和 unlink()函数。

【例 11.1】 复制、重命名和删除文件。

```
<?php
    fopen('abc.txt', 'w');                    //创建文件 abc.txt

    if(!file_exists('abc.txt'))               //判断文件 abc.txt 是否存在
    {
        echo "the specified file does not exist.";
                        //若不存在，输出“指定文件不存在”的提示
    } else {
        if (!file_exists("new_abc.txt"))//判断文件 new_abc.txt 是否存在
        {
            copy('abc.txt','new_abc.txt');
                        //若不存在，则将文件 abc.txt 复制到 new_abc.txt 中
        } else {
            echo 'The target file already exists. The COPY is terminated.';
        }
```

```
        if (!file_exists('abc_new.txt'))//判断文件 abc_new.txt 是否存在
        {
            rename('new_abc.txt','abc_new.txt');
                                        //若不存在，则将文件重命名为 new_abc.txt
        } else {
            echo "There is already a file with the new name specified.
            The RENAME is terminated";
        }
    }

    if (file_exists('abc.txt'))         //判断文件 abc.txt 是否存在
    {
        unlink('abc.txt');              //若存在，则删除文件 abc.txt
    } else {
        echo "The file does not exist. The DELETION is terminated.";
    }
?>
```

在上面这段脚本中，我们首先使用 fopen()函数创建了一个可读写的文本文件 abc.txt。然后使用 file_exists()函数来判断文件 abc.txt 是否创建。如果文件已创建，则使用 copy()函数将其复制一份至 new_abc.txt，再使用 rename()函数将文件 new_abc.txt 重命名为 abc_new.txt。最后，使用 unlink()函数删除最先创建的 abc.txt 文件。

运行该脚本后，浏览器里不会出现任何信息，而当前文件夹下则多出了一个名为 abc_new.txt 的文本文件。

11.1.3　组织文件

在操作系统中，文件是通过文件夹进行整理的。在本小节里，我们就来看看如何修建和移除文件夹，以及如何获取某文件夹中的文件和文件夹列表。

如果需要创建一个文件夹，我们可以使用 mkdir()函数。如：

```
mkdir("testdir");
```

该语句会在当前路径下创建一个名为“testdir”的文件夹。若当前路径下已存在同名的文件夹，则 PHP 会显示如下告警：

```
Warning: mkdir("testdir"): File exists in c: on line 2.
```

所以，在创建文件前，应使用 is_dir()函数来判断一下同名文件夹是否存在。如：

```
if(!is_dir("testdir"))
{
    mkdir("testdir");
} else {
    echo "The specified directory already exists.";
}
```

如果需要在某个文件夹下创建一个子文件夹，可以使用绝对路径，也可以使用相对路径。如：

```
mkdir("/root/nextdir/mynewdir");        //绝对路径
mkdir("../mynewdir");                   //相对路径
```

在 Windows 操作系统中，所谓的绝对路径是指带上盘符的路径，如“C:\nextdir\mynewdir”；而相对路径则是指以脚本文件所在位置为参照点的路径。若脚本文件所在位置的绝对路径为“C:\nextdir”，而我们需要在 nextdir 文件夹下创建子文件夹 mynewdir，则该子文件夹的相对路径为“..\mynewdir”。其中的“..”指的是上一级目录，“\”后的内容指的是下一级目录。

在 Linux 操作系统中，不存在盘符的说法，根目录为 root 目录。所以，绝对路径都是以 root 目录为基准的；而相对路径则是以脚本文件所在的位置为基准。需要注意的是，在 Linux 操作系统中，目录与子目录之间是用“/”分隔的，与 Windows 操作系统不同。

如果需要从当前文件夹切换到另一个文件夹，则可以使用 chdir()函数。所需参数与 mkdir()相同，都是文件夹的路径。如：

```
chdir("..\mynewdir");
```

如果需要列出指定文件夹中的所有文件名，可以先用 opendir()函数和 readdir()函数将指定文件夹读取到一个变量中，接着使用 while 循环输出指定文件夹下的所有文件名。其中，opendir()函数可以看作是一个游标，它返回的是当前游标指针指向的一条记录；而 readdir()函数是从指针指向的记录读取出文件名。

我们可以使用如下的脚本输出上一章的例程目录中的文件名列表。

【例 11.2】 输出指定目录下的文件名列表。

```
<?php

    $dh = opendir("..\chapter10");      //将指定目录读取到一个游标变量中

    $filename = readdir($dh);
                                //将游标指针指向的文件的文件名读取到一个文本变量中

    while($filename = readdir($dh))
                                //若当前游标指向不是列表末尾，则输出该文件对应的文件名
    {
        echo $filename."<br>";
    }
?>
```

11.2　调用操作系统命令

在 Windows 操作系统中，我们可以通过命令行窗口向操作系统发出各种各样的指令来完成相应的操作。如果我们想查看某个文件夹中的文件列表，可以使用 dir 命令；如果想要复制某个文件到指定的位置，可以使用 copy 命令；如果想要创建文件夹，可以使用 mkdir 命令。

在本节中，我们将介绍如何使用 PHP 来调用操作系统的命令完成相应的操作。完成本节的内容需要对 Windows 操作系统的 MS-DOS 命令有一定的了解。虽然 PHP 提供了很多的原生函数帮助对操作系统进行相应的操作，但是仍旧有些操作是 PHP 力有不逮的。比如，我们想复制某个指定的文件夹，包括其中的子文件夹和文件，或者想在 PHP 脚本中执行使用其他编程语言写出的程序，但 PHP 没有提供相应的原生函数，这时，就只能在 PHP 脚本中嵌入与这些操作相关的指令或程序。

PHP 为我们提供了四种方法来调用操作系统命令和执行使用其他编程语言写出的程序。

- 重音符（`）：PHP 可以执行嵌入到一对重音符（`）之间的操作系统命令并返回，使命令输出。这一对重音符其实是 shell_exec()函数的简写形式。
- system()函数：使用 PHP 的 system()函数可以执行一条操作系统命令并返回命令的最后一行输出。
- exec()函数：使用 PHP 的 exec()函数可以执行一条操作系统命令并将命令输出存入指定数组，然后返回命令的最后一行输出。
- passthru()函数：使用 PHP 的 passthru()函数可以执行一条操作系统命令并返回命令输出。

值得注意的是，如果需要使用这四项函数来执行系统命令，需要确认它们在 php.ini 配置文件中没有被禁用。如需开启这四项函数，可以打开 php.ini，查找“disable_functions =”关键字，然后将等号右边的 shell_exec、system、exec 和 passthru 都去掉。在保存修改后的 php.ini 之后，重启 apache 服务器以便修改生效。

11.2.1　重音符（`）

执行系统命令最简单的一种方法，就是把需要执行的命令放在一对重音符之间，然后将其赋给某个变量。如果需要输出该命令的结果，则可以使用 echo 或者 print 之类的打印命令字进行输出。

例如，我们想要输出 C 盘下一个名为 python27 文件夹里的文件列表，则可以运行如下脚本：

```
$result = `dir c:\python27`;
```

当 PHP 执行了这一句脚本之后，变量$result 中就保存了系统命令“dir c:\python27”的执行结果，也就是一串文件夹和文件列表。这时，我们可以使用 echo 命令输出执行结果到浏览器中，得到类似如下的结果：

```
驱动器 C 中的卷没有标签。
 卷的序列号是 E8A0-76E7

 c:\python27 的目录

2013-06-12  19:08    <DIR>          .
2013-06-12  19:08    <DIR>          ..
2013-06-12  19:08    <DIR>          DLLs
2013-06-12  19:08    <DIR>          Doc
2013-06-12  19:08    <DIR>          include
2013-06-12  19:08    <DIR>          Lib
2013-06-12  19:08    <DIR>          libs
2013-05-15  22:53            40,098 LICENSE.txt
2013-05-15  22:41           359,882 NEWS.txt
2013-05-15  22:43            26,624 python.exe
2013-05-15  22:43            27,136 pythonw.exe
2013-05-15  22:41            54,979 README.txt
2013-06-12  19:08    <DIR>          tcl
2013-06-12  19:08    <DIR>          Tools
2013-05-15  22:43            49,664 w9xpopen.exe
               6 个文件        558,383 字节
```

```
                9 个目录 25,699,856,384 可用字节
```

11.2.2　system()函数、exec()函数和 passthru()函数

除此之外，我们还可以使用 system()函数来调用系统命令，输出命令执行结果。与前面提到的那一对重音符不同的是，当使用 system()函数调用某系统命令时，系统就会自动将该命令的执行结果输出到浏览器。如果我们将 system()函数的返回值赋给某个变量，并将该变量的值输出到浏览器，那么将看到命令执行结果的最后一句打印了两次。因为 system()函数返回的值是该命令执行结果的最后一行。

exec()函数与 system()函数的返回值一样，都是在执行命令后返回该命令执行结果的最后一行。但是 exec()函数与 system()函数不同的是，exec()函数不会自动输出命令执行结果到浏览器。因此，当使用 exec()函数将某条系统命令的执行结果赋值给某个变量并将该变量的值输出到浏览器，我们只会看到该命令执行结果的最后一行。如果需要查看命令执行结果的其他内容，可以将通过 exec()函数执行的命令的执行结果保存到一个数组中，然后通过 foreach 命令字输出命令的执行结果：

```
$result = exec("dir c:\python27", $dirout)

foreach($dirout as $line)
{
    echo $line. "<br>";
}
```

上面这段脚本中，$dirout 是我们定义的数组，exec()函数将命令“dir c:\python27”的结果输出到这个数组中；然后使用 foreach 遍历这个数组，得到与之前那一对重音符相同的结果。当然，我们也可以使用如下的脚本输出指定行的内容：

```
echo $dirout[3];
echo $dirout[7];
```

与 system()和 exec()函数都不同的是，passthru()函数不会返回任何值，只会自动输出命令执行结果到浏览器。因此，大家不要用 passthru()函数来为变量赋值，要不然在输出变量时得到的只能是空值。

11.2.3　四个变量的区别

下面我们比较一下这四个变量，如表 11-2 所示。

表 11-2　shell_exec()、system()、exec()和passthru()对比

	函数功能	返回值	返回值内容	其他输出
shell_exec()	调用系统命令，输出命令执行结果	有	字符串，命令执行结果	无
system()		有	字符串，命令执行结果的最后一行	自动输出全部执行结果至浏览器
exec()		有	字符串，命令执行结果的最后一行	无
passthru()		无	无	自动输出全部执行结果至浏览器

我们提到过，在使用这些函数之前，需要确认它们在 php.ini 配置文件中有没有被禁用。因为通过这些函数，恶意用户可能会执行你不想让他们执行的命令或者获取你不想授予他们的权限。因此，在不了解是否可能会造成安全风险之前，建议禁用这四个函数。

11.3　使用 PHP 操控 FTP

在互联网上，每天都会有成千上万的文件从一台电脑传送到另一台电脑。当一家公司不同分部的员工们想要通过互联网传送文件，那真是一点儿问题也没有。现如今，我们可以使用的文件传输方式真是数不胜数，方式虽多，但是不外乎两种方式，一种是通过 Web 服务器，也就是通过超文本传输协议（HTTP），另一种则是通过文件传输协议（FTP）。在之前的章节里，我们学习了如何通过 PHP 调用 HTTP 上传文件到服务器。现在来看看如何使用 PHP 调用 FTP 软件来传输文件。

FTP 软件通常都是一分为二的客户端/服务器架构。使用 FTP 软件在本地电脑和远程电脑之间传输文件，需要在本地电脑安装 FTP 软件的客户端，在远程电脑上安装 FTP 软件的服务器端。然后通过客户端和服务器端通信在本地电脑和远程电脑间建立 FTP 连接，从而实现文件传输。

若需要通过 PHP 脚本使用 FTP 协议，则需要在安装了 PHP 引擎之后，启用 FTP 支持。在 Windows 操作系统中，FTP 支持是默认开户的。但在 Linux、UNIX 及 Mac 操作系统中，FTP 支持需要手动开启。

学习本章的内容，需要架设 FTP 服务器。互联网上开源并免费的 FTP 服务器有很多，这里我们使用 FileZilla 服务器。需要下载的读者，可以访问 FileZilla 项目的官方网站（https://filezilla-project.org/）获取。

11.3.1　准备工作

在安装好 FileZilla 之后，双击桌面上的 FileZilla 服务器的图标，启动 FileZilla 服务器。这时，如图 11-1 所示的 FileZilla 的管理界面也显示在了桌面上。

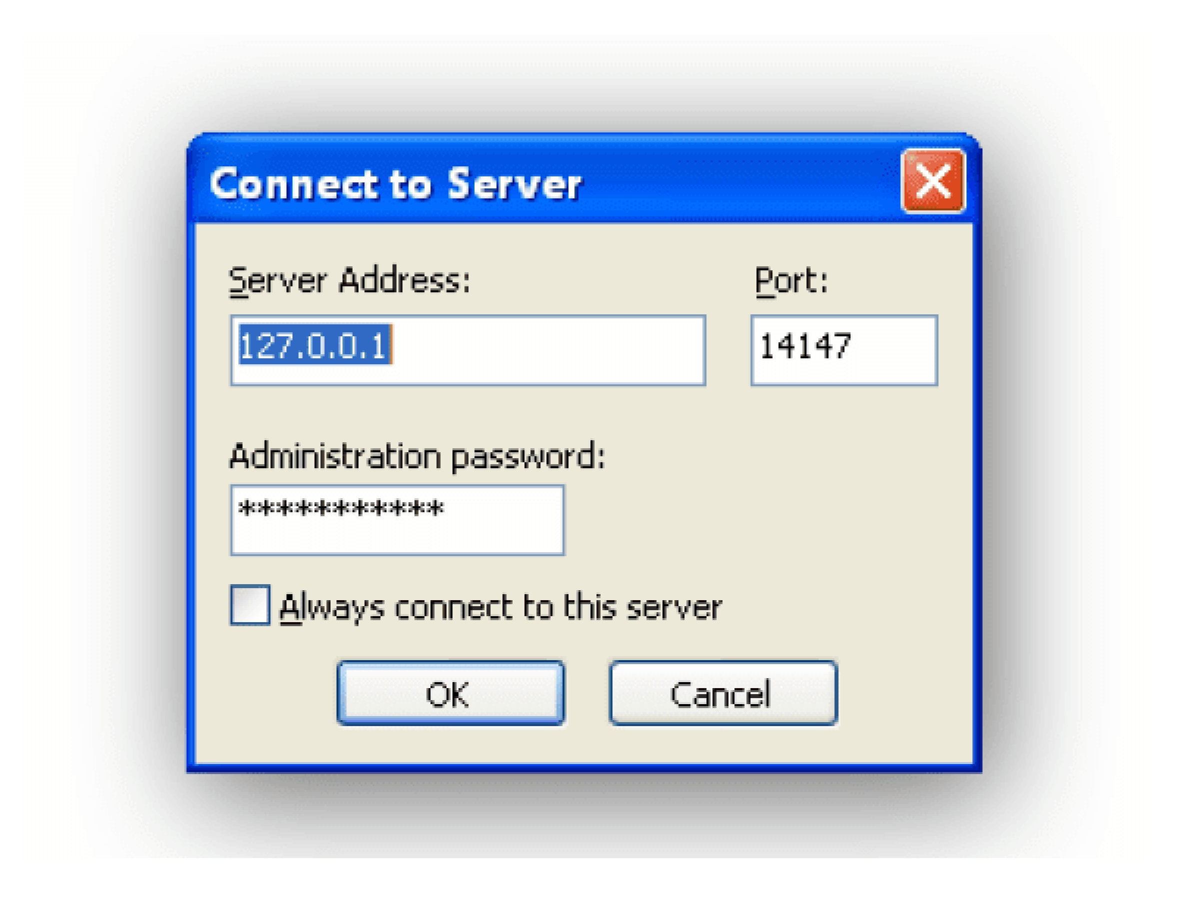


图 11-1　FileZilla 管理登录界面

通过这个管理登录界面，我们可以知道服务器的地址（Server Address）是 127.0.0.1。后面的端口号（port）是用来对 FTP 服务器进行管理时使用的专用端口号，不可向外界透露。管理密码（Administration password）则需要自行指定。在完成这一切后，单击 OK 按钮连接上 FileZilla 服务器。

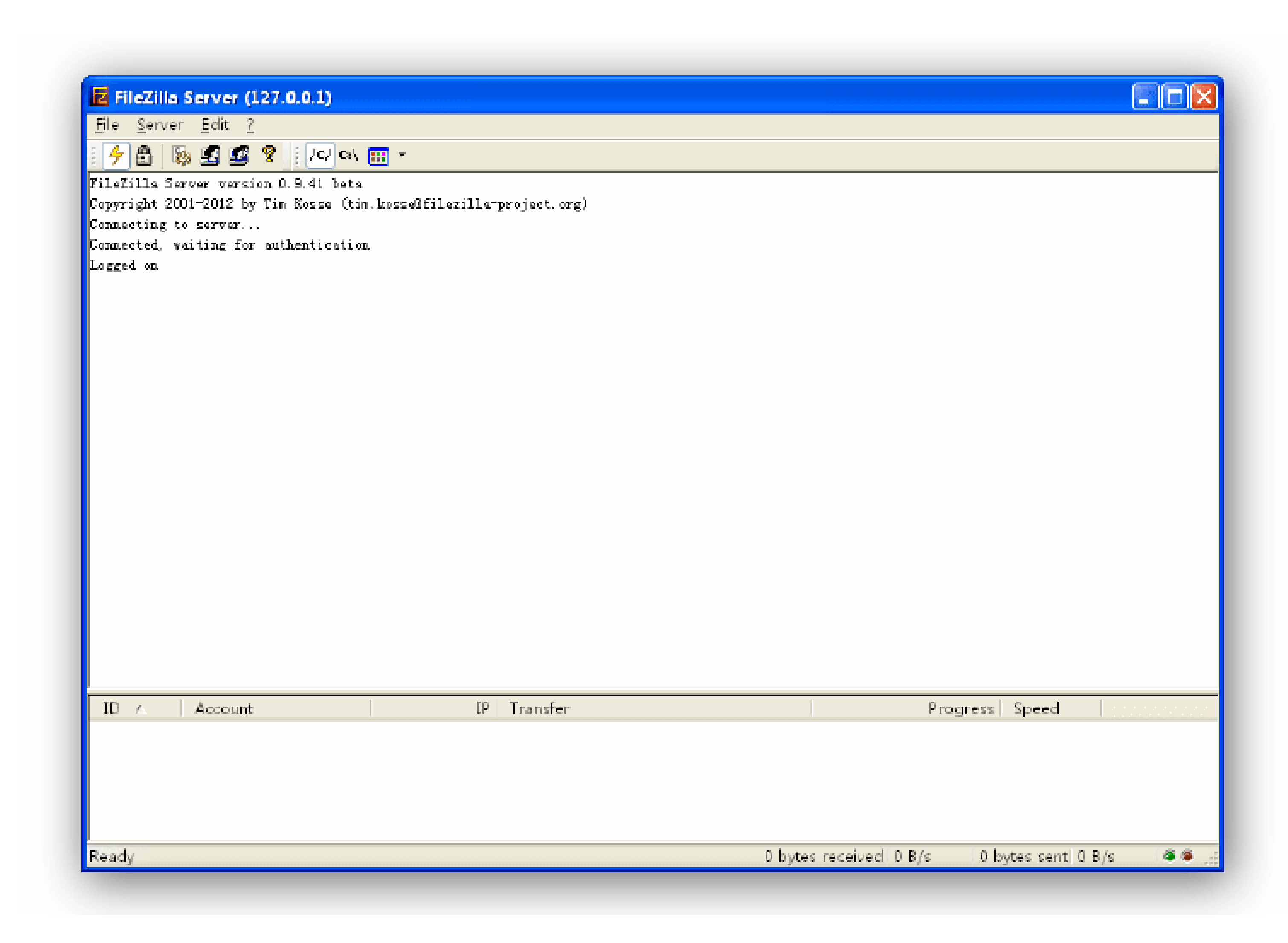


图 11-2　FileZilla 服务器管理界面

在连接上 FileZilla 服务器后，进入了 FileZilla 服务器的管理界面，如图 11-2 所示。在这里我们可以对服务器进行管理和配置。

先在服务器上建立一个 FTP 用户，并为其分配一个可访问的目录。

（1）单击工具栏上的“ ”按钮，弹出“用户”对话框，如图 11-3 所示。

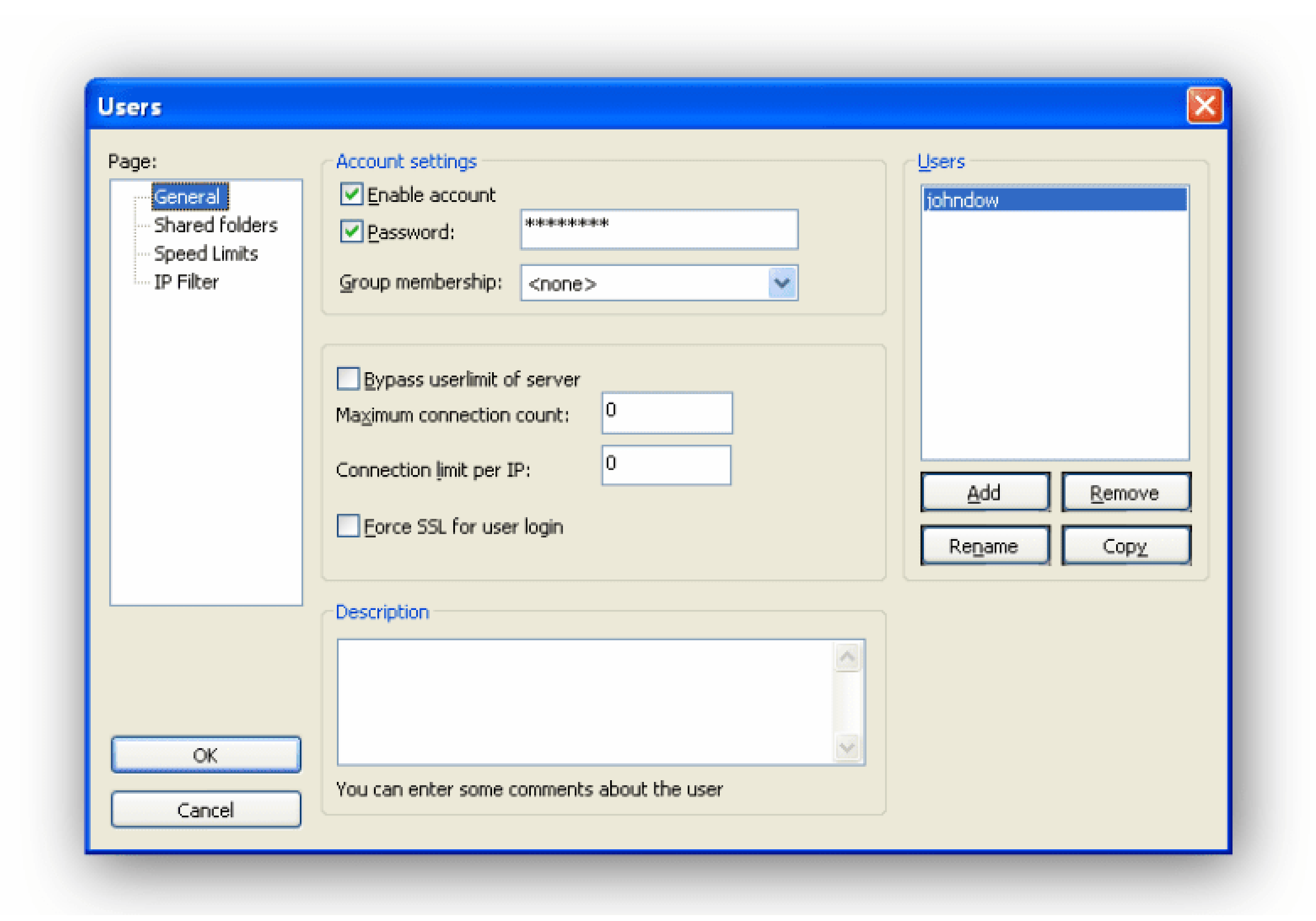


图 11-3　FileZilla 服务器用户管理界面

（2）在用户管理界面右侧的“用户（Users）”区域框里单击 Add 按钮。在弹出的窗口中输出用户名 johndow，分组信息为默认的“<none>”。然后，单击“确认”按钮。

（3）在“用户（Users）”区域框右侧的“账户设置（Account settings）”区域框内勾选密码（Password），并在其右侧的文本框里为上一步中添加的用户指定密码“Password*”。

（4）单击用户管理界面左侧的页面区域框里的“Shared folders”页面。然后单击在其右侧出现的“Shared folders”区域框内的“Add”按钮，为用户名 johndow 的用户分配共享文件夹，并为其分配适当的文件操作权限。然后单击窗口左下角的 OK 保存设置并关闭用户管理界面，如图 11-4 所示。

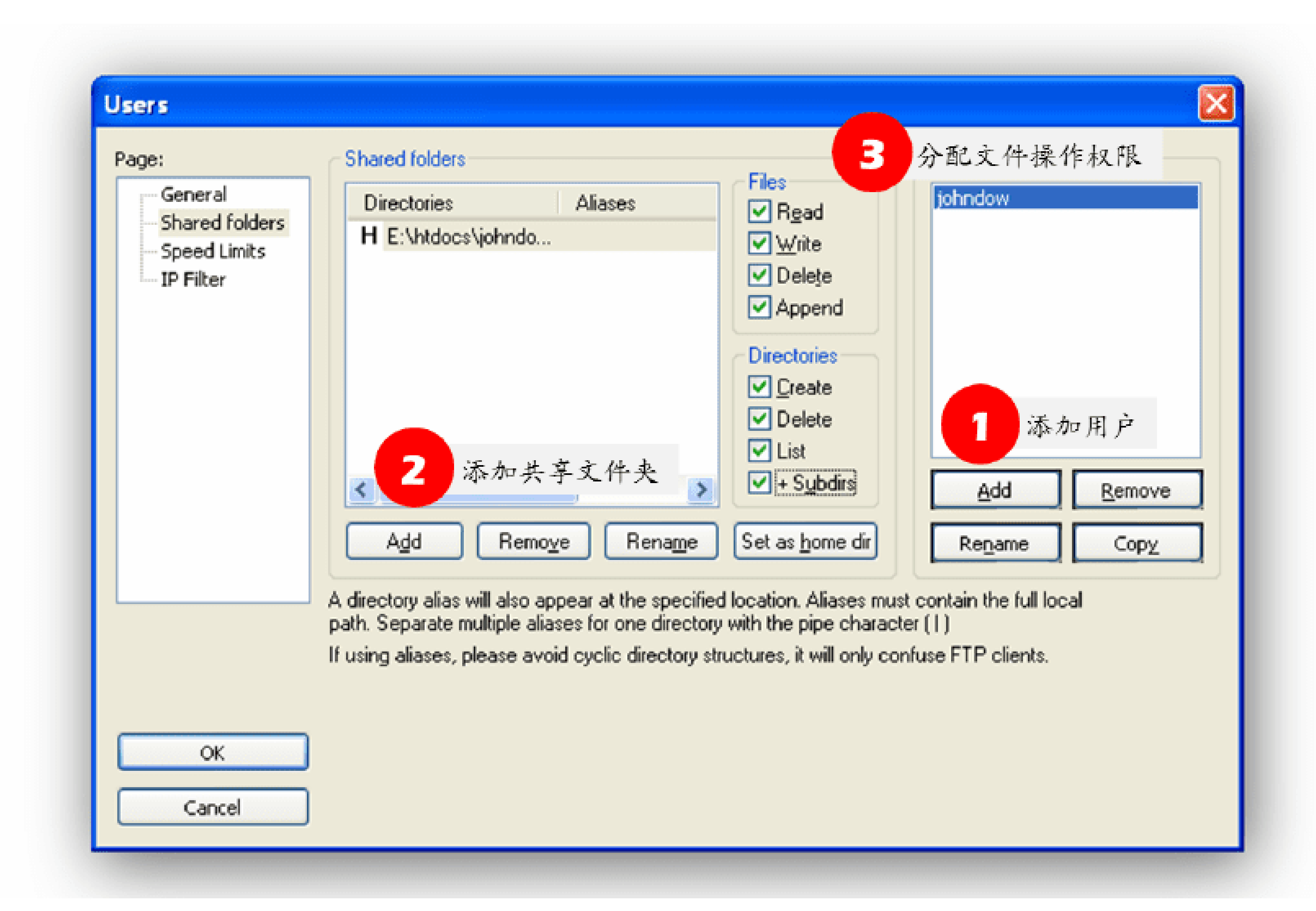


图 11-4　为用户 johndow 设置共享文件夹并分配文件操作权限

至此，我们的准备工作就做好了。

11.3.2　登录 FTP 服务器

在登录 FileZilla 服务器之前，需要在本地电脑和 FTP 服务器间建立一条 FTP 连接。在 PHP 脚本中，可以使用 ftp_connect()函数来实现：

```
$connect = ftp_connect('127.0.0.1');
```

如果我们为 FileZilla 服务器配置了域名的话，ftp_connect()的参数也可以是 FileZilla 服务器的域名。如：

```
$connect = ftp_connect('www.example.com');
```

执行该脚本，系统没有提示任何错误，则表明 FTP 连接成功建立了。

接下来，我们需要向 FTP 服务器提交用户名和密码用于验证身份。这时就需要使用 ftp_login()函数，该函数有三个必要参数，分别是 FTP 连接对象、用户名和密码。

```
$login_result = ftp_login($connect, 'johndow','Passw0rd*');
```

如果我们在 FTP 连接没有建立的情况下执行 ftp_login()函数，系统会发出警告，同时继续执行余下的脚本。为了防止余下的脚本不会在 FTP 连接没有建立的情况下执行，我们可以把建立连接的脚本修改为：

```
$connect = ftp_connect('127.0.0.1') or die('找不到服务器。');
```

或者

```
$connect = ftp_connect('www.example.com') or die('找不到服务器。');
```

同样地，为了防止余下的脚本不会在没有成功登录的情况下执行，我们也可以在登录服务器的脚本修改为：

```
$login_result = ftp_login($connect, 'johndow', 'Passw0rd*') or die('无法登录服务器。');
```

当我们执行登录脚本后，若系统没有发生任何告警或错误提示，表明成功登录了 FileZilla 服务器。现在我们就可以获取服务器上的文件列表、向服务器上传文件或从服务器下载文件了。

11.3.3　获取服务器文件列表

在 FTP 服务器上最常见的操作是获取文件列表。为此，PHP 为我们准备了 ftp_nlist()函数来实现这项操作。该函数有两个必要参数，分别为 FTP 连接对象和需获取文件列表对应的文件夹名。在这里，我们需要列出分配给“johndow”用户的“johndow_filezilla_ftp”文件夹中的文件列表，脚本如下：

```
$filesArr = ftp_nlist($connect, 'johndow_filezilla_ftp');
```

这时，指定文件夹中的文件列表就以数组的形式存入到了$filesArr 中。

如果我们并不知道分配给 johndow 用户的文件夹叫什么名字，则可以使用 ftp_pwd()函数来获取当前文件夹。当以 johndow 用户的身份登录服务器时，当前文件夹就是之前我们分配 johndow 用户的文件夹。因此，上一句脚本可以修改为：

```
$directory = ftp_pwd($connect);
$filesArr = ftp_nlist($connect,$directory);
```

之后，我们就可以像遍历普通数组的方式输出当前文件夹下的文件列表：

```
foreach ($filesArr as $value)
{
    echo $value.'<br>';
}
```

11.3.4　下载和上传文件

我们可以使用 ftp_get()函数从 FileZilla 服务器下载文件。该函数有四个参数，分别是 FTP 连接对象、文件下载到本地后的文件名、文件在 FileZilla 服务器上的文件名以及文件类型。在这里我们需要将 FileZilla 服务器上的 file01.txt 文件下载到本地，并以 newfile.txt 的文件名进行保存，则脚本可以写成：

```
ftp_get($connect, 'newfile.txt', 'file01.txt', FTP_ASCII);
```

对于 ftp_get()函数的最后一个参数的“文件类型”只有两种，一种为 FTP_ASCII，另一种为 FTP_BINARY。前者指的是如文本文件之类的可以不需借助其他阅读工具就可以阅读的文件，而后者则指的是余下的其他类型的文件。这里我们需要下载的是 file01.txt 文件，因此，文件类型应该指定为 FTP_ASCII。

在运行了上面的脚本之后，若 file01.txt 文件确实存在，则会被下载到本地目录中，并被重命名为 newfile.txt。

与从服务器下载文件类似，向服务器上传文件需要用到 ftp_put()函数。该函数同样也有四个参数，分别是 FTP 连接对象、文件上传后在服务器上的文件名、文件在本地的文件名及文件类型。在这里我们需要将本地的 newfile.txt 上传到 FileZilla 服务器上，则脚本可以写成：

```
ftp_put($connect, 'newfile.txt', 'newfile.txt', FTP_ASCII);
```

若本地确实存在一个名为 newfile.txt 的文件，则在脚本执行后，该文件会被上传到 FileZilla 服务器上并以 newfile.txt 的名称进行存储。

11.3.5　使用 PHP 操控 FTP

需要注意的是，在完成了对 FileZilla 服务器的操作之后，需要使用 ftp_close()函数断开 FTP 连接。该函数只有一个参数，那就是 FTP 连接对象。

现在我们把本节学习的内容总结一下。

【例 11.3】 使用 PHP 操控 FTP。

```
<?php
    $connect = ftp_connect("127.0.0.1")             //连接服务器
                or die("找不到服务器");
    $login_result = ftp_login($connect, 'johndow', 'Passw0rd*')
                                                    //登录服务器
                or die("无法登录服务器");

    $directory = ftp_pwd($connect);                 //获取当前目录
    $filesArrBefore = ftp_nlist($connect, $directory);
                                            //获取当前目录的文件列表

    fileList($filesArrBefore);              //输出当前目录的文件列表

    ftp_get($connect, 'newfile.txt', 'file01.txt', FTP_ASCII);
                                            //下载 file01.txt 到本地

    ftp_put($connect, 'newfile.txt', 'newfile.txt', FTP_ASCII);
                                        //上传 newfile.txt 文件到服务器

    echo '<hr>';

    $filesArrAfter = ftp_nlist($connect, $directory);
// 再次获取当前目录的文件列表

    fileList($filesArrAfter);               //输出当前目录的文件列表

    ftp_close($connect);                    //关闭当前 FTP 连接
```

```

    function fileList($filesArr)            //遍历函数
    {
        foreach ($filesArr as $value)
        {
            echo $value."<br>";
        }
    }
?>
```

在运行了例 11.3 中的脚本之后，可以发现浏览器中列出了两份文件列表，其中后一份列表比前一份多出了一个文件 newfile.txt。而本地和服务器的目录中则分别多出了两个名为 newfile.txt 的文件。

11.4　使用 PHP 发送电子邮件

当今电子邮件是使用最为广泛的互联网应用之一。许多的 PHP 应用程序都会要求发送电子邮件到用户邮件来验证用户身份；或者向用户发送订单确认信息等。例如，在许多的网站上会有“忘记密码”这个链接，单击这个链接，输入相应的信息。不久后，就会收到一封电子邮件帮助找回丢失的密码。电子邮件的应用可谓层出不穷。那么我们如何才能使用 PHP 来发送电子邮件和附件呢？

PHP 提供了一些函数，可以让我们简便地发送电子邮件。在本节里，我们将了解如何使用 PHP 脚本来发送电子邮件和附件。

11.4.1　准备工作

在使用 PHP 发送电子邮件之前，我们需要搭建邮件服务器。与 FTP 服务器类似，网络上也有许多开源的邮件服务器。在这里我们使用国产的 Magic Winmail 服务器来演示如何在本地搭建和配置邮件服务器。更多的内容可以参见 Magic Winmail 的官方网站（http://www.magicwinmail.com/docs/）支撑文档。

在下载并安装好了 Magic Winmail 服务器后，启用 Winmail 服务器程序。当系统托盘处出现图标时，表示邮件服务器已经启用。首次启用邮件服务器时会出现设置向导，关闭即可。双击桌面上的 Magic Winmail 管理端工具图标，弹出如图 11-5 所示的“连接服务器”窗口。



图 11-5　Magic Winmail 管理端工具登录窗口

（1）在“登录用户”区域框的“密码”处输入在安装过程中设置的管理密码，然后单击“确定”按钮登录 Winmail 邮件服务器。成功登录服务器后弹出图 11-6 所示的对话框。

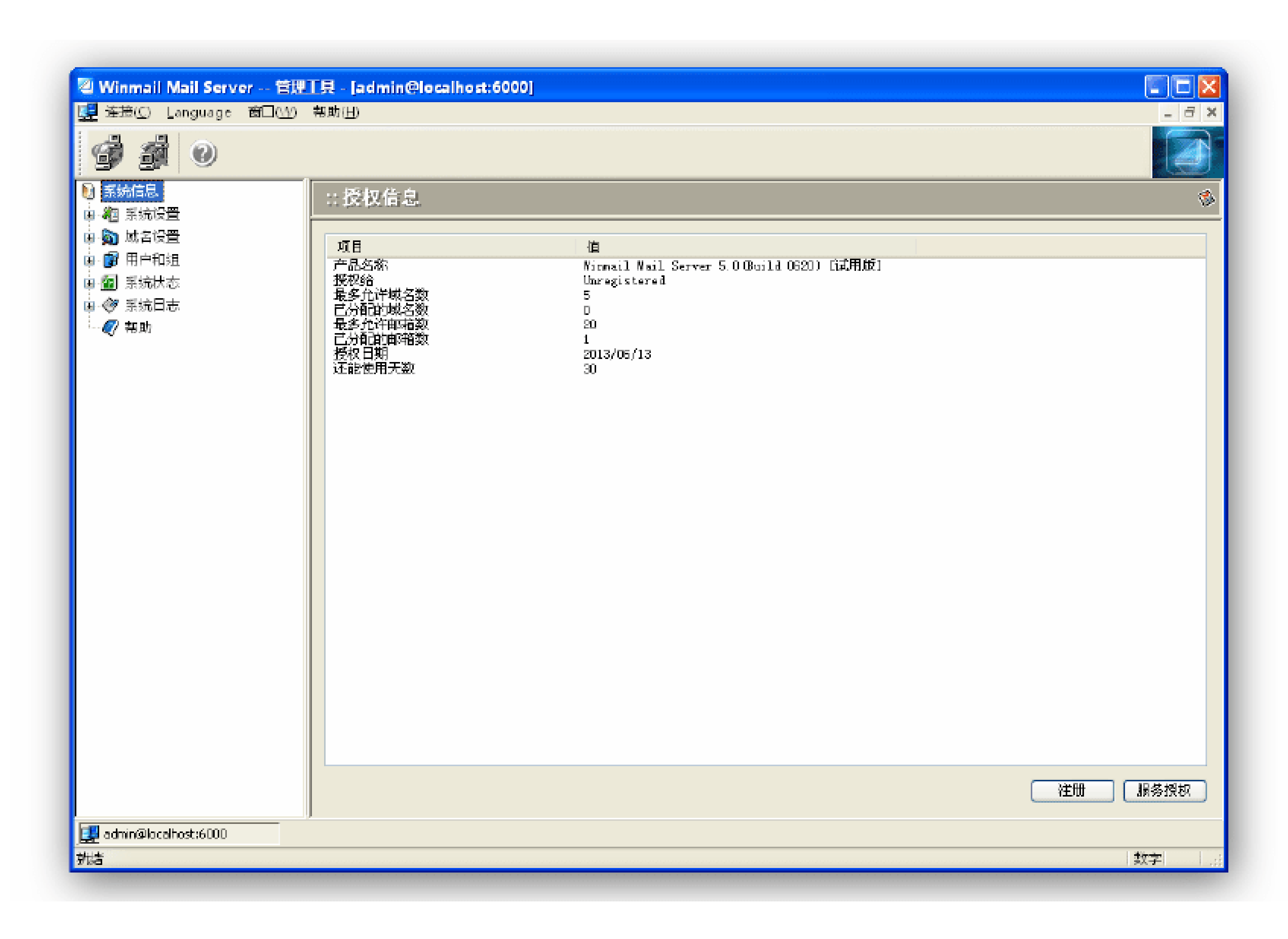


图 11-6 成功登录 Magic Winmail 服务器管理端工具

（2）展开左树的“域名设置”，单击“域名管理”子项为邮件服务器添加主域。在单击了“域名管理”页面左下角的“新增”按钮之后，弹出“域名”窗口。这时，在“基本参数”页签下的“域名”文本框里输入“localhost.com”，然后单击“确定”按钮。至此，服务器的主域添加完成，如图 11-7 所示。

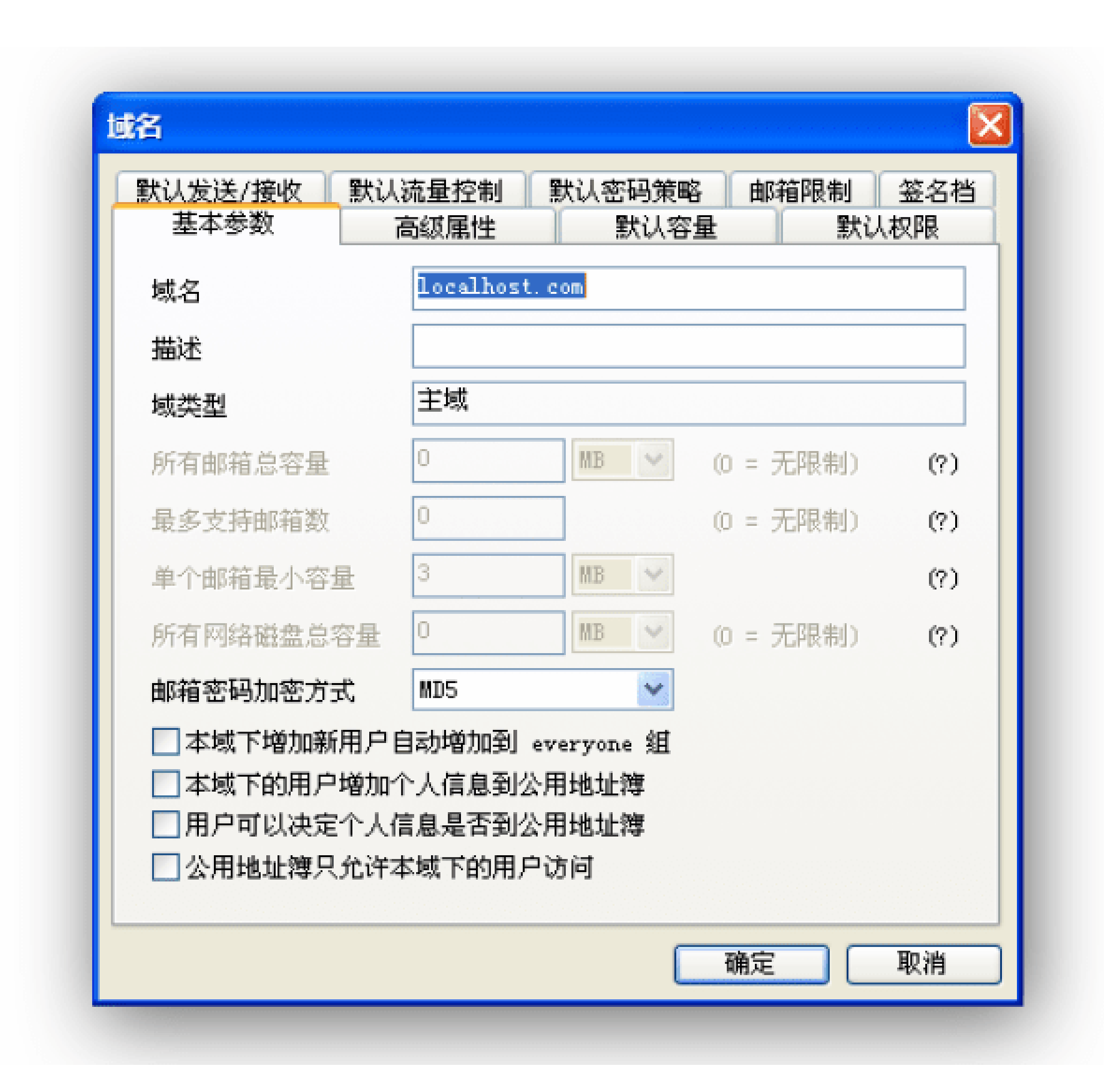


图 11-7 为邮件服务器添加主域

（3）展开左树的“用户和组”，单击“用户管理”子项添加用户。系统中已经默认存在一个管理员用户 postmaster。单击“用户管理”页面左下角的“新增”按钮，在弹出的“基本设置”对话框内输入用户信息，然后单击“完成”按钮。这时“用户管理”页面上列

出了刚才添加的用户，如图 11-8 所示。

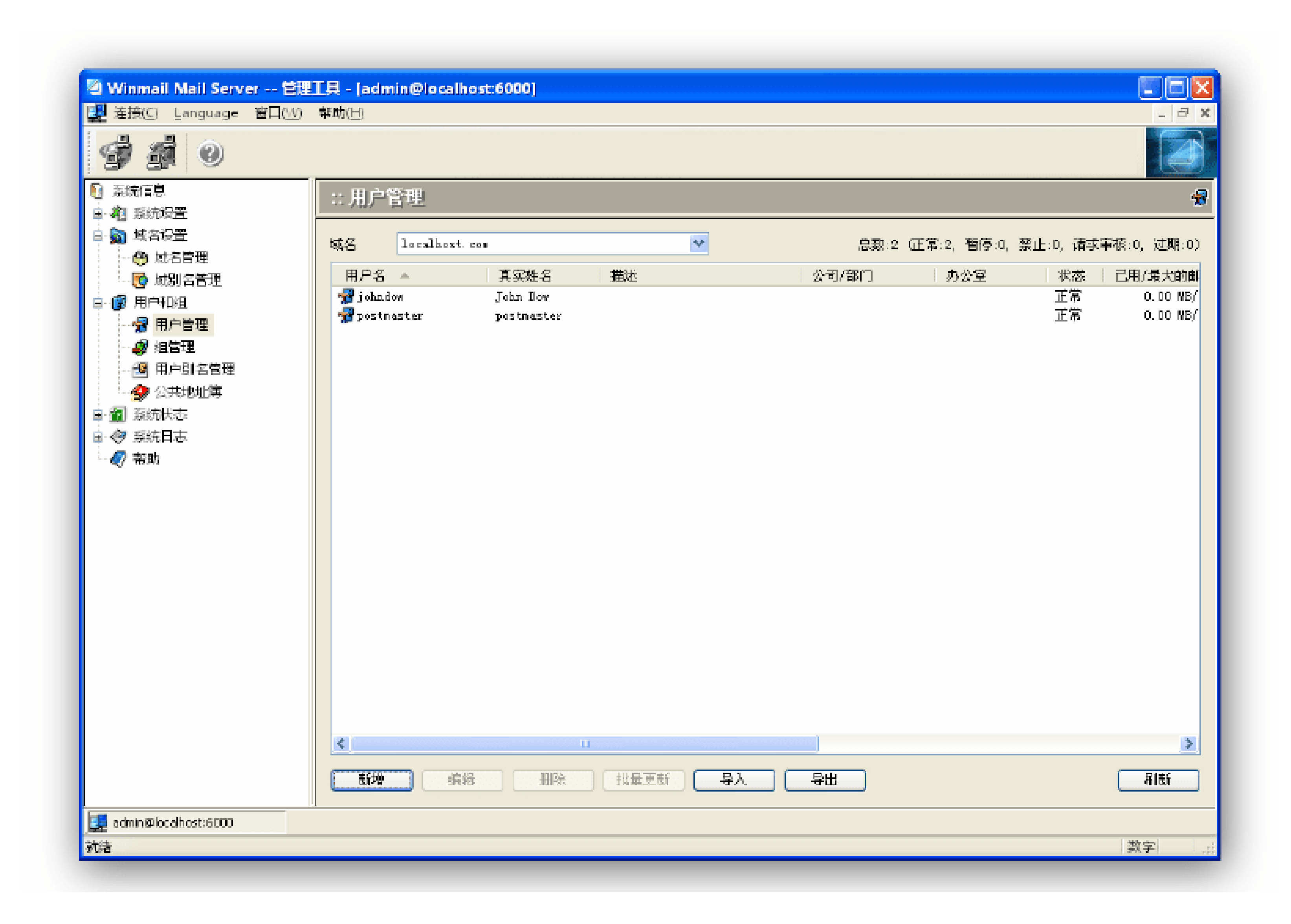


图 11-8　添加用户

（4）然后打开 PHP 引擎的 php.ini 文件，找到“[mail function]”。将其项下的“SMTP”设置为“localhost”，“sendmail_from”设置为“postmaster@localhost.com”。然后检查一下这两项参数前面是否有分号（;），若有，则删除这些分号，如图 11-9 所示。随后，重启 Apache 服务器。

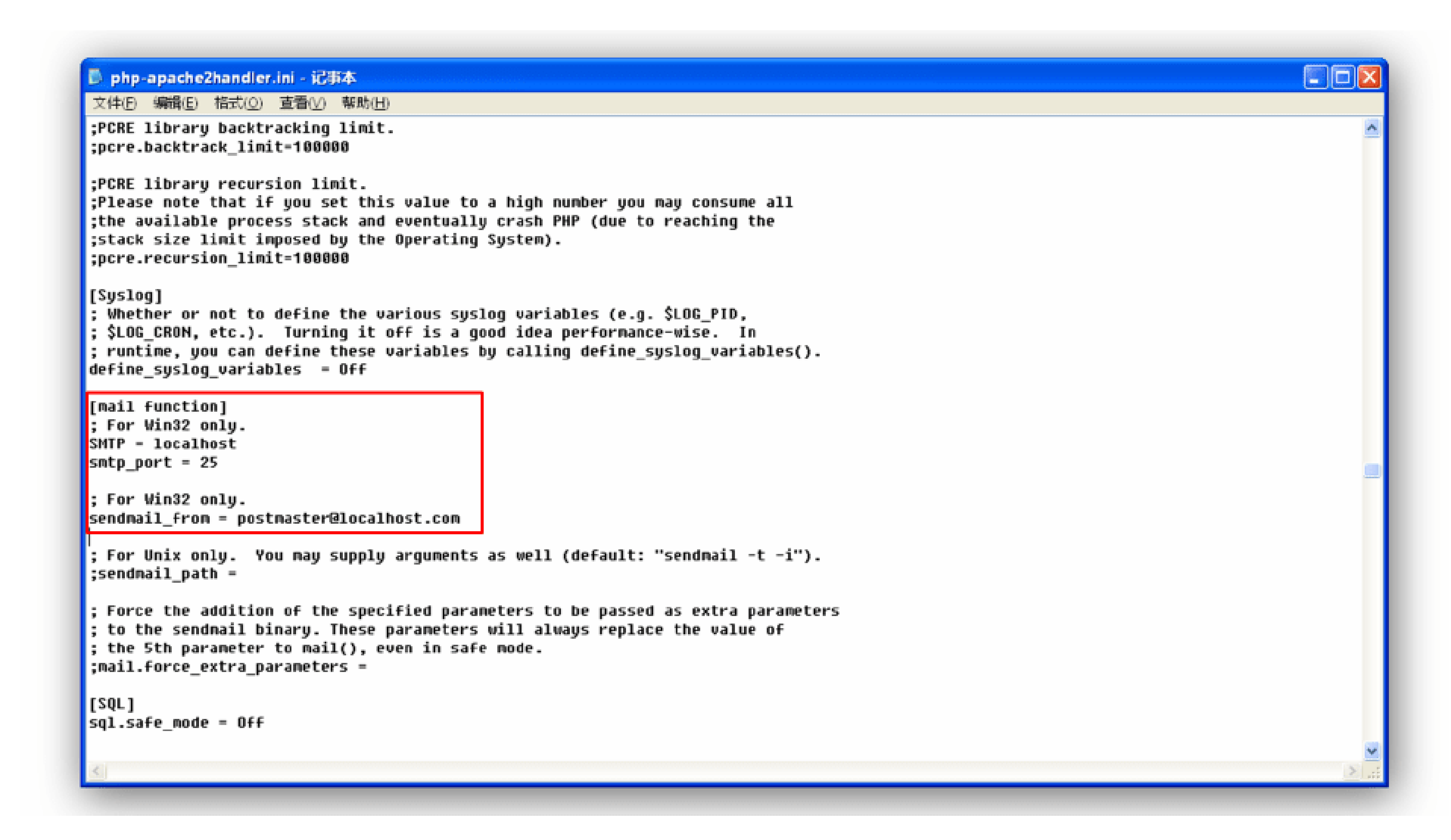


图 11-9　修改 php.ini

（5）现在打开浏览器，在地址框内输入“http://127.0.0.1:6080”，弹出 Magic Winmail 的 Webmail 界面。然后输出之前添加的用户 John Dow 的用户名和密码，单击“登录>>>”按钮。

在 Webmail 的首页，我们可以看到 John Dow 用户的邮箱内有一封未读邮件。记住这个数字，因为待会儿将使用 PHP 脚本向 John Dow 的邮件再发送一封邮件。若到时，John Dow

的邮箱内有两封未读邮件，则表示我们通过 PHP 脚本向 John Dow 发送邮件成功了。

11.4.2　发送电子邮件

在 PHP 脚本中，我们可以使用 mail()函数向指定用户发送邮件。这个函数有三个必选参数和一个可选参数。格式如下：

```
mail(address, subject, message, headers);
```

这四个参数的说明如下：

- address 指的是收件人的邮箱地址。在这里就是 John Dow 的邮箱地址。
- subject 指的是邮件的主题。
- message 指的是邮件的内容。
- headers 指的是邮件头信息。

在这四个参数中，前三个参数是必选的，第四个参数是可选的。

现在我们就来看看如何使用 PHP 脚本向 John Dow 的电子邮箱发送一封邮件。

【例 11.4】 使用 mail()函数向指定用户的电子邮箱发送电子邮件。

```
<?php
    $to = "johndow@localhost.com";
    $subj = "test";
    $mess = "This is a test of the mail function";
    $headers = "bcc:techsupport@localhost.com\r\n";
    $mailsend = mail($to,$subj,$mess,$headers);
?>
```

在上面的这段脚本中设置了四个变量，分别为$to、$subj、$mess 和$headers。这四个参数分别对应收件人地址、邮件主题、邮件内容和邮件头信息。最后一句使用了 mail()函数实现了邮件的发送。

运行了上面这段脚本后，我们再登录 John Dow 的 Webmail 邮箱。查看一下 John Dow 的收件箱里是不是收了一封来自 postmaster@localhost.com 的邮件呢？如图 11-10 所示。

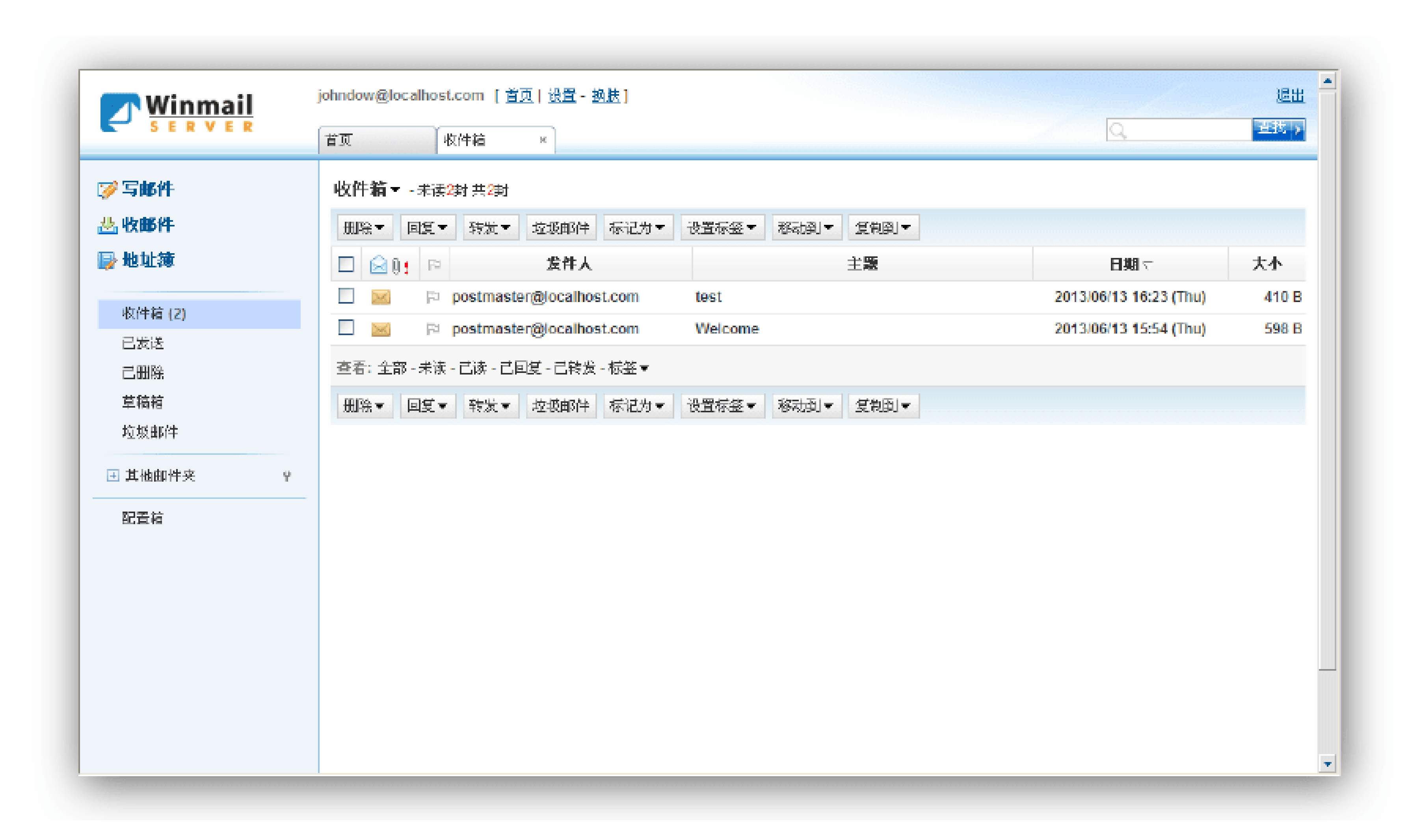


图 11-10　检查用户的收件箱，确认发送结果

我们惊喜地发现，John Dow 的收件箱里的确多了一条来自 postmaster@localhost.com 的主题为“test”邮件。说明，我们使用 PHP 脚本发送电子邮件成功了。

11.4.3　发送带附件的电子邮件

现如今，大家都喜欢在邮件里发个附件。那么使用 PHP 脚本发送带附件的电子邮件要如何做呢？这里假设要在邮件里带上一张动画图片，具体如何实现请看下面的解析。

首先，我们先来看一份电子邮件在传送过程中的样子：

```
Return-Path:
Date: Mon, 22 May 2000 19:17:29 +0000
From: Someone
To: Person
Message-id: <83729KI93LI9214@example.com>
Content-type: multipart/mixed; boundary="396d983d6b89a"
Subject: Here's the subject
--396d983d6b89a                                    //邮件头信息到此结束
Content-type: text/plain; charset=iso-8859-1
Content-transfer-encoding: 8bit

This is the body of the email.

--396d983d6b89a                                    //邮件内容到此结束
Content-type: text/html; name=attachment.html
Content-disposition: inline; filename=attachment.html
Content-transfer-encoding: 8bit

This is the attached HTML file


--396d983d6b89a—                                   //附件内容到此结束
```

在这封邮件中，前七行为邮件头信息。在邮件头信息中，我们发现 Content-type 为 multipart/mixed，而 boundary 为一随机字符串。这一随机字符串做为分隔邮件头和邮件主体、以及邮件主体中的邮件内容和附件内容的分隔符。注意该随机字符串出现了三次，其中，第一次和第二次出现时，其前面加上了“--”符号，而最后一次出现时，其前后都加上了“--”。

如果需要发送带附件的邮件，则需要分别构造邮件头信息和邮件主体。

【例 11.5】 使用 mail()函数向指定用户发送带附件的电子邮件。

```
<?php
//定义收信人
$to = 'johndow@localhost.com';
//定义邮件主题
$subject = 'Test email with an image attached';                #1
//创建邮件部件边界
//在这里，我们使用 md5()函数，以当前时间为对象生成一个唯一随机数
$random hash = md5(date('r', time()));                          #2
//定义需要传送的邮件头信息。注意邮件头之间用“\r\n”隔开
$headers = "From: podmaster@localhost.com\r\nReply-To: podmaster@
localhost.com";
//添加邮件部件边界和 MIME 类型
$headers .= "\r\nContent-Type: multipart/mixed; boundary=\"PHP-mixed-".
```

```
$random_hash."\"";                                              #3
//读取附件文件内容到一个字符串变量
//用 base64_encode()函数对该字符串变量进行编码
//然后将编码后的内容分割成若干小块以利传输
$attachment = chunk_split(base64_encode(file_get_contents('mouth.gif')));
                                                                #4
//定义邮件内容
ob_start(); // 打出输出至缓存选项                                #5
?>

--PHP-mixed-<?php echo $random_hash; ?>
Content-Type: multipart/alternative; boundary="PHP-alt-<?php echo
$random_hash; ?>"                                               #6

--PHP-alt-<?php echo $random_hash; ?>
Content-Type: text/plain; charset="iso-8859-1"                  #7
Content-Transfer-Encoding: 7bit

Hello World!!!
This is simple text email message.

--PHP-alt-<?php echo $random_hash; ?>
Content-Type: text/html; charset="iso-8859-1"                   #8
Content-Transfer-Encoding: 7bit

<h2>Hello World!</h2>
<p>This is something with <b>HTML</b> formatting.</p>

--PHP-alt-<?php echo $random_hash; ?>--                         #9

--PHP-mixed-<?php echo $random_hash; ?>                         #10
Content-Type: image/gif; name="mouth.gif"
Content-Transfer-Encoding: base64
Content-Disposition: attachment

<?php echo $attachment; ?>
--PHP-mixed-<?php echo $random_hash; ?>--                       #11

<?php
//将当前缓存内容存入到变量$message 中，然后清空缓存
$message = ob_get_clean();                                      #12
//发送电子邮件
$mail_sent = @mail( $to, $subject, $message, $headers );        #13
//若邮件发送成功，显示“Mail sent”。反之则显示“Mail failed.”
echo $mail_sent ? "Mail sent" : "Mail failed";
?>
```

运行上面这段脚本之后，我们再次登录 John Dow 的 Webmail 邮箱，发现其收件箱中多出了一封名为“Test email with an image attached”的邮件。打开该邮件，赫然发现邮件正文和附近都发送成功了。而且附件还以签名的形式显示在正文的下方，如图 11-11 所示。

上面这段脚本一共由两对“<?php … ?>”标记对分隔开的三部分组成。在第一对“<?php … ?>”标记对中，定义了收件人、邮件主题和邮件头，创建了邮件部件边界，读取附件文件内容并对其进行编码；在第二对“<?php … ?>”标记对中，定义了邮件正文的内容并发送了邮件，在这两对“<?php … ?>”标记对之间，我们撰写了邮件的正文和附件。按照顺序把这三部分的内容分别叫做：

- ❑ 邮件起始部分；
- ❑ 邮件正文与附件；
- ❑ 邮件发送部分。

需要注意的是，在脚本中使用 ob_start()开启了“输出至缓存”选项。因此，所有的内容都会被直接发送到缓存中存储，不再经过脚本处理。如果 ob_start()之后的各行内容前有空格或者不相干的字符，这些空格和不相干的字符也会被送进缓存，进而破坏邮件内容，导致在邮件中看不到相应的内容。另外，在需要缓存的内容处理完成后，务必使用 ob_get_clean()函数获取缓存的数据并清空缓存。

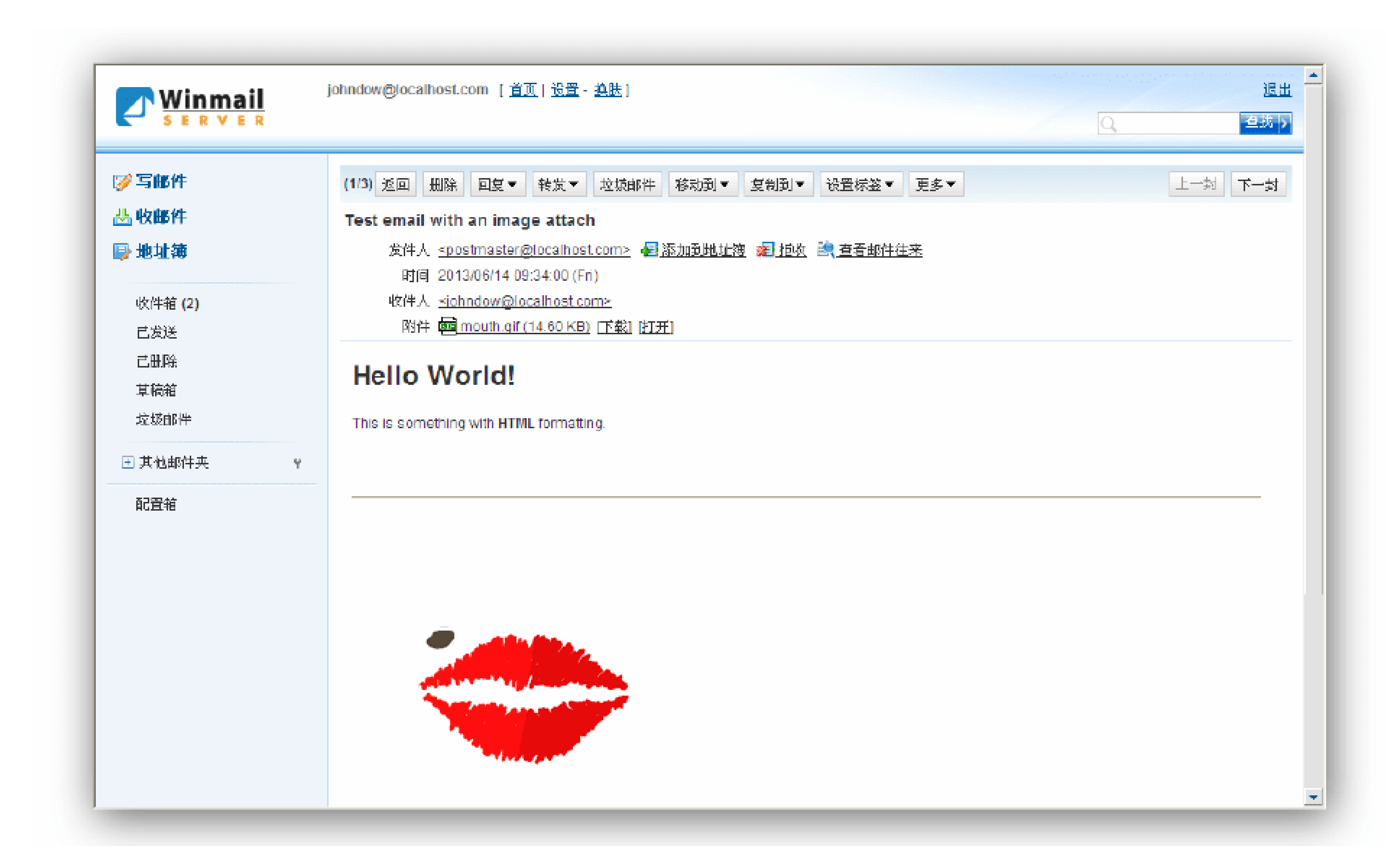


图 11-11　收到通过 PHP 发送的带附件的电子邮件

现在我们就来一步一步地了解如何构造一封带附件的电子邮件。

第一步　构建邮件的起始部分

在这一部分里，我们重点关注例 11.5 所示脚本中的标记为“#1”～“#5”的内容。

- ❑ 标记为“#1”的行与其上一行定义了收件人和邮件主题。
- ❑ 标记为“#2”的行定义了一个随机数，以供构造邮件部件边界。
- ❑ 标记为“#3”的行及其上一行定义了邮件头。

注意，邮件头信息有许多种。在这两行里，我们看的邮件头字段有“From”、“Reply-To”、“Content-Type”和“boundary”。前三个字段的字段名与字段值之间用的是冒号（:），而“boundary”字段的字段名与字段值之间用的则是等号（=）。由于在这封邮件中既有正文也有附件，所以这里的 Content-Type 应该定义为“multipart/mixed”，表示邮件是多个部件混合而成。在这段脚本中，字段名与字段值之间使用等号连接的邮件头字段有“boundary”、“charset”和“name”，分别代表邮件部件边界、字符集和附件文件名。

- ❑ 标记为“#4”的行读取了需要传送的附件，并对其进行了编码和分块处理。

这里使用到了三个函数，分别是 file_get_contents()、base64_encode()和 chunk_split()。函数 file_get_contents()将附件的内容读取到一个字符串中，然后交由 base64_encode()函数进行编码，编码后的内容再交由 chunk_split()函数进行分块，最后得到$attachment 字符串。

- 标记为“#5”的行开启了 PHP 引擎的“输出至缓存”选项。

开启该选项后，其后的内容将会被直接送至缓存。

第二步　构建邮件的正文和附件

在这一部分里，我们需要构建的正文和邮件的附件都会以字符串的形式送进缓存。

- 标记为“#6”的行往下至第 38 行为邮件的正文。

在这一部分里，我们构建了两个可选的子部分。若邮件服务器支持 HTML 代码的显示，则以 HTML 代码显示正文，否则以纯文本的形式显示正文。因此，在标记为“#6”的行中我们定义的“Content-Type”为“multipart/alternative”表示往下的内容为可选，同时也定义了一个新的边界，用来分隔可选内容。

- 标记为“#7”和“#8”的行分别定义了两个可选的部分。

前者为以纯文本方式显示正文的内容，所以其“Content-Type”为“text/plain”，而后者则以 HTML 代码的方式显示正文的内容，所以其“Content-Type”为“text/html”。在这两个可选的部分里，我们使用的字符集都是 iso-8859-1，向下兼容 ASCII。另外，在这两个部分里，我们还使用了一个新的字段“Content-Transfer-Encoding”用来定义 chunk_split() 函数分块的大小。

- 标记为“#9”的行定义了一个可选部分边界，并以“--”结尾。标志着再无其他可选部件。
- 标记为“#10”的行定义了一个邮件部件边界，开始构建另一个邮件部件。

从该行始往下至第 45 行为邮件附件的内容。由于附件文件为一张 GIF 图片，因此“Content-Type”为“image/gif”，同时还定义了一个新的字段“name”，供邮件服务器获取附件文件的名称。由于之前我们使用 base64_encode()函数对读取的附件文件内容进行编码，因此，“Content-Transfer-Encoding”为“base64”。在这一部分里，我们还定义了一个新的字段“Content-Disposition”，该字段的值告诉邮件服务器如何处理当前部件的内容。这里我们将“Content-Disposition”定义为“attachment”，就是告诉邮件服务器将当前部分做为邮件附件进行处理。

- 标记为“#11”的行定义了一个邮件部分边界，并以“--”结尾，标志着再无其他邮件部件。邮件内容（包括正文和附件）到此结束。

至此，邮件正文和附件构建完毕。需要再次提醒注意的是，写入缓存的各行内容前不得有空格和其他的字符。如果有的话，这些空格和字符也会被送入缓存，破坏邮件内容，造成邮件正文和附件无法正常显示。

第三步　构建邮件发送部分

在这一部分里，我们需要从缓存中获取邮件的内容（包括正文和附件），然后使用 mail()函数发送邮件。

- 标记为“#12”的行使用了 ob_get_clean()函数从缓存中获取数据并清空缓存。从缓存中获取的数据被存放在$message 变量中。
- 标记为“#13”的行使用了 mail()函数发送了邮件。若邮件发送成功，则显示“Mail sent”；若失败，则显示“Mail failed”。

第 12 章　PHP 与基于对象的编程（OOP）

正如我们在之前的章节中看到的那样，PHP 允许用两种不同的风格编写脚本：一种是基于过程的脚本编写风格（procedure programing），而另一种则是基于对象的脚本编写风格（object-oriented programming）。读者可以选择其中的一种风格开始编写脚本。当然，如果愿意的话，也可以在基于对象的脚本中引入基于过程的脚本或者在基于过程的脚本中编入基于对象的脚本。

在本书的前面几章中，大多数的例程都是基于过程的脚本。但是，在第 8 章、第 9 章和第 10 章的例程和实战练习里，我们使用的又是基于对象的脚本编写风格。如果读者还是不清楚两者的区别，我们来看看下面两段脚本。

【例 12.1】 基于过程的编程风格与基于对象的编程风格。

```
<?php
    //基于过程的编程风格
    echo '<h3>基于过程的编程风格</h3>';
    $a = 5;
    $b = 6;

    echo '<h4>a = '.$a.', b= '.$b.'</h4>';
    echo 'a + b = '.($a + $b).'<br>';
    echo 'a - b = '.($a - $b).'<br>';
    echo 'a x b = '.($a * $b).'<br>';
    echo 'a / b = '.($a / $b).'<br>';

    $a = 10;
    $b = 15;

    echo '<h4>a = '.$a.', b= '.$b.'</h4>';
    echo 'a + b = '.($a + $b).'<br>';
    echo 'a - b = '.($a - $b).'<br>';
    echo 'a x b = '.($a * $b).'<br>';
    echo 'a / b = '.($a / $b).'<br>';
```

在上面这段脚本中，可以看到我们先定义了两个变量，然后使用 echo 命令一行一行地输出了它们自身的值以及这两个变量的和差积商；接着我们又重新定义了这两个变量的值，然后再一次地输出了这两个变量在重新定义后的值及它们的和差积商。看看脚本中加粗的部分，发现所有的计算都是在行间即时完成的。

现在再来看一段使用基于对象的编程风格实现相同功能的脚本：

```
    include_once('class/class.calc.php');

    //基于对象的编程风格
    $calc = new calc();

    echo '<h3>基于过程的编程风格</h3>';
```

```
    echo '<h4>a = '.$calc->getVarA().', b= '.$calc->getVarB().'</h4>';

    echo 'a + b = '.$calc->add().'<br>';
    echo 'a - b = '.$calc->minus().'<br>';
    echo 'a x b = '.$calc->multiply().'<br>';
    echo 'a / b = '.$calc->divide().'<br>';

    //使用 calc 实例的 setVarA 和 setVarB 方法来设置私有变量的值
    $calc->setVarA(10);
    $calc->setVarB(15);

    echo '<h4>a = '.$calc->getVarA().', b= '.$calc->getVarB().'</h4>';
    echo 'a + b = '.$calc->add().'<br>';
    echo 'a - b = '.$calc->minus().'<br>';
    echo 'a x b = '.$calc->multiply().'<br>';
    echo 'a / b = '.$calc->divide().'<br>';

?>
```

在比较两段脚本之后，可以发现第二段脚本除了在加粗的部分上与第一段脚本不同之外，其他都是一样的，并且第二段脚本中加粗的部分比第一段脚本中加粗的部分看上去要复杂的多。首先，我们定义了一个 calc 类的实例，并将其命名为$calc；然后使用$calc 实例的 getVarA 和 getVarB 方法获取了 calc 类的内置变量$a 和$b 的值，接着使用$calc 实例的 add、minus、muliply 和 divide 方法输出这两个内置变量的和差积商。接着，使用$calc 实例的 setVarA 和 setVarB 方法重新设置了变量$a 和$b 的值，然后再次使用 add、minus、mutiply 和 divide 方法输出这两个内置变量的和差积商。所有的计算都是通过 calc 类的公有方法实现的。

这段脚本虽然看上去复杂，但是阅读起来似乎比第一段清晰，至少通过阅读代码可以很清楚的知道每一块的意思是什么。可能有的读者认为第一段脚本直来直去的更加清晰，那是因为这两段脚本实现的功能都很简单。当我们需要编写复杂的程序时，会发现基于对象的编程风格可以让脚本的可读性大大提升。这也是为什么我们要学习基于对象编程的一个重要的原因。

12.1　基于过程与基于对象

12.1.1　为什么要用 OOP

基于对象的编程让开发人员的日子变得好过了许多。而正是因为基于对象的编程可以把我们碰到的问题变成一个一个更加细小的问题从而使问题更加容易理解、容易解决。

基于对象编程的主要目标是通过对象完成所有需要完成的任务。对象可以说是基于对象编程中最小、最基本的部件。我们可以通过在对象间共享数据来解决问题。

一般来说，使用基于对象编程的好处主要有如下几个方面。

1. 可读性

一个对象做为一个实体拥有一系统列的属性和方法，同时也可以和其他的对象互动共

同完成某个任务。有时候，一个对象可以独立存在，就像在本章的前言中看到的$calc 对象，而有的对象则需要依附于其他的对象。通常，一个对象用于解决一类具体的问题，就像我们定义的$calc 对象，可以用来设置和获取 calc 类的内置变量，同时使用其内置的方法计算内置变量的和差积商。如果在开发过程中碰到了一系列有联系的问题，那么可以尝试定义一个解决这些问题的类，然后再通过实例化该类来解决这些问题。

2．重用性

如果同时开发几个 PHP 项目，而这些项目有着某些相同点，那么使用基于对象的编程可以方便地将某一个项目中使用的类移植到另一个项目中，从而实现类的重用。这样一来，就不用在每一个项目中重新编写脚本了。

3．扩展性

如果想要在开发的项目中方便地添加新的方法和属性，使用基于对象的编程是不二的选择。基于对象的编程最核心的功能就是扩展性。可以重构对象来为其添加新的方法或属性，同时也可以继续保持该对象的向下兼容能力，兼容原有的脚本。另外，还可以重新创建一个保留了其父本对象中必要方法和属性的新对象，然后向其中添加新的方法和属性。这种特性也叫做继承性，我们在第 8 章涉及过这个概念。

4．可维护性

基于对象的脚本比基于过程的脚本维护起来要容易的多。因为基于对象的脚本遵循了某种严格的编码规范，通过明白的逻辑结构实现需要的功能。打个比方，如果使用基于对象的脚本，那么可以方便地对开发的脚本进行扩展，也可以将其分散成若干个片段分发给和你开发同一项目的其他开发人员。这些分散开来的片段也可以成为独立的对象完成特定的任务，大家互不影响、独立开发。需要整合的时候，也可以非常容易地把这些分散的片段整合在一起。

5．高效性

使用基于对象的编程可以提升脚本的执行效率和简化脚本开发流程。有许多既定的开发模式可以供我们开发更好更高效的脚本。

基于上述原因，我们可以使用基于对象的编程方式来提升代码的可读性、重用性、扩展性、可维护性和代码的执行效率。使我们以更加合理的开发模式将任务化整为零，步步为营地将其解决。

12.1.2　对象面面观

在第 8 章里学习了类和对象的概念，知道如何定义一个类，如何通过实例化类来创建对象以及如何使用对象来完成特定的任务。在本小节里，以例 12.1 中引用的 calc 类为例来解剖一个对象。详细了解一下什么是对象、如何使用对象。

【例 12.2】 Calc 类的使用。

```
class calc {
    private $a;
    private $b;

    //为私有变量设置初始值
    function __construct(){
        $this -> a = 5;
        $this -> b = 6;
    }

    //设置私有变量$a 和$b;
    function setVarA($num){
        $this -> a = $num;
        return $this -> a;
    }

    function setVarB($num){
        $this -> b = $num;
        return $this -> b;
    }

    //获取私有变量$a 和$b;
    function getVarA(){
        return $this -> a;
    }

    function getVarB(){
        return $this -> b;
    }

    //计算变量$a 和$b 的和;
    function add(){
        $result = $this -> a + $this -> b;
        return $result;
    }

    //计算变量$a 和$b 的差;
    function minus(){
        $result = $this -> a - $this -> b;
        return $result;
    }

    //计算变量$a 和$b 的积;
    function multiply(){
        $result = $this -> a * $this -> b;
        return $result;
    }

    //计算变量$a 和$b 的商;
    function divide(){
        $result = $this -> a / $this -> b;
        return $result;
    }
}
```

看完上面这个类之后，我们可以清楚地知道，这个类有两个私有属性和八个公有方法。其中四个方法用来设置和获取这两个私有属性的值，而剩余的四个方法则用来计算这两个私有属性的和差积商。如果想要使用这个类，首先需要实例化这个类，然后使用经过实例

化的对象来使用上述八个公有方法操作对象的私有属性。

那么到底什么是对象呢？说白了，对象就是一段定义了若干属性和方法的脚本。它和数组很像，一个数组可以将若干个值定义在不同的键中，而一个对象则拥有若干个属性和方法。它与数组不同的是，对象中的属性和方法并不都是可以在实例化后使用的。比如例 12.2 中的 calc 类里定义的两个私有属性在 calc 类实例化后是不能直接访问的。通常情况下，一个类在实例化后，我们只能访问其公有发生和方法。

现在，再仔细地阅读一下例 12.2 中的 calc 类，我相信这段脚本在你眼中会变得越来越好懂。如果你也使用这种风格编写脚本的话，会慢慢地发现基于对象编写脚本的好处。从而编写的脚本也会变得越来越容易管理和维护。

在 Wordpress 的官方网站的首页下方，可以发现一句话“Coding is Poetry”（编码如写诗）。只要施得其法，编码的确就像写诗一样。

12.1.3　基于对象编程中常用术语

在本小节里，我们会列出在基于对象编程中经常会用到的一些术语。其中有一些在第 8 章里学习过，这里就当复习一下。了解这些术语以及它们背后的意义对于我们继续学习基于对象的编程是十分有益的。

1. 类

类可以被看做是对象的模板，类定义了对象可以使用的方法和属性。基于某个类的对象只能在类规定的范围内活动和与其他的对象互动。每当创建了一个基于某个类的对象，这个对象就成为这个类的实例。

2. 属性

属性是在类中定义的，用来存放一些信息的变量。值得注意的是，属性只能在类中定义，并被基于类的对象或子类使用。在一个类中定义的属性可以有三种适用范围，一些只能在定义它们的类中使用，称为私有属性；一些可以在定义它们的类以及继承自该类的子类中使用，称为保护属性；还有一些则可以被所有人使用，称为公有属性。在类的方法中定义的变量不是类的属性，这些变量只能在这些函数中使用。

3. 方法

方法是在类中定义的，用来执行某些操作的函数。它与属性类似，据其适用范围的不同，分为私有方法、保护方法和公有方法。

4. 封装

所谓封装是将某脚本（方法）和使用该脚本（方法）操控的数据绑定在一起的一种机制。把它们封装在一起，可以有效地避免它们受到外界的干扰。把脚本（方法）和受脚本（方法）操控的数据绑定到类的过程称为封装。类的封装性使得开发人员不用再担心类中各方法的实现原理和过程。

5．多态

对象的状态是多变的。一个对象相对于同一个类的另一个对象来说，它们拥有的属性和方法虽然相同，但却可以有着不同的状态。另外，一个类可以派生出若干个子类，这些子类在保留了父对象的某些属性和方法的同时，也可以定义一些新的方法和属性，甚至于完全改写父类中的某些已有的方法，这一过程称为类多态化。

6．继承

继承是类多态化的诸多方法中最典型的一种。当定义一个继承自另一个类的子类时，子类可以继承父类的所有方法和属性，也可以改写其中的某些方法和属性，还可以添加一些新的方法和属性。

7．耦合

耦合指的是类之间的依赖程度。使用轻耦合类间架构的脚本相较于使用重耦合类间架构的脚本可用性要更高。在下一节里，我们会了解到关于耦合这个概念的更多细节。

8．模式

这里的模式特指设计模式，在基于对象的编程中，它常常用来解决一系列相似的问题。使用特定的设计模式，可以用极少的脚本改写提升脚本的执行效率。当然，特定的设计模式是把双刃剑，如果在不需要它的地方使用了它，则会降低脚本的执行效率。

9．子类

子类是指从某个类中派生出来的类。通常拥有父类的某些属性和方法，同时也有一些自有的属性和方法，还有可能拥有一些改写过的父类的属性和方法。

10．父类

父类是指派生出子类的类。

11．实例

当创建一个基于某个类的对象时，就实例化了这个类。而这个被创建的对象就是该类的一个实例。

12.1.4　基于对象编程的编码规范

在本章的代码里，我们会遵循一些编码规范。这些规范虽然不甚严格，但也足以使脚本可读和可维护。另外，在遵循这些规范的同时，也可以避免引入一些冗余的对象，从而提升脚本的执行效率。

现在一起来看看这些规范：

- 在一个.php 文件里只定义一个类。在这个类之外，不使用基于过程的编码风格。
- 类和存放类的文件要有合适的名字。比如在例 12.2 中定义的类，就应该放在名为

class.calc.php 文件中并放在名为 class 的文件夹下。这样做的好处是，可以很方便地在一堆文件中找到想要找的文件，而不用费时费力地一个文件一个文件地查看。

- 在为类的文件命名时，都使用小写字母。
- 与命令一般类和类文件一样，对于存放接口类的文件定义成 interface.name.php，保存在名为 interface 的文件夹下；对于存放抽象类的文件则定义成 abstract.name.php，保存在名为 abstract 的文件夹下；对于存放 Final 类的文件则定义成 final.name.php，保存在名为 final 的文件夹下。
- 用驼峰命令法来命名类，如果一个类的名字只含有一个英文单词，则全部小写，如 calc；如果一个类的名字中含有多个英文单词，则从第二个单词开始，每个单词的首字母大写，如 myCalc。
- 在类中定义属性和方法时，也请务必使用驼峰命令法。

12.2　初识 OOP

在正式开始编写基于对象的代码之前，我们需要复习一下在第 8 章里学习到的一些基本概念，包括类、对象、继承、改写、修饰和魔术方法等。

12.2.1　类和对象

在第 8 章学习了类和对象的概念。在这一小节里，我们来复习与类和对象相关的一些基本概念。

首先还是来看一段脚本。

【例 12.3】 创建名为 notes 的类（\\class.notes.php）。

```
<?php
   //创建一个名为 notes 的类
   class notes {
       //定义一些属性
       private $noteDate;
       private $weather;
       private $author;
       private $filename;

       function __construct($author,$weather){
           //为私有属性赋初始值
           $this -> setNoteDate(date('Y 年 m 月 d 日'));
           $this -> setWeather($weather);
           $this -> setAuthor($author);
           $this -> setFileName($this -> author);


           //初始化存储日志的文本文件
           $this -> initStorage();
       }

       //配置存储日志的文本文件
       private function initStorage(){
```

```
        $filename = $this -> filename;
        if(!file_exists($filename)){
            $fh = fopen($filename, 'w');
            fwrite($fh, "----start----\r\n");
            fclose($fh);
        }
    }

    //定义设置日志作者、天气和日期的方法
    private function setFileName($author){
        $this -> filename = 'file/'.base64_encode($author).'.notes';

    }

    private function setAuthor($author){
        $this -> author = $author;
    }

    private function setWeather($weather){
        $this -> weather = $weather;
    }

    private function setNoteDate($noteDate){
        $this -> noteDate = $noteDate;
    }

    //定义获取日志作者、天气和日期的方法
    protected function getFileName(){
        return $this -> filename;
    }

    function getAuthor(){
        return $this -> author;
    }

    function getWeather(){
        return $this -> weather;
    }

    function getNoteDate(){
        return $this -> noteDate;
    }

    //定义获取所有日志和指定日志的方法
    function getNotes(){
        $filename = $this -> filename;

        $fh = fopen($filename, 'r');
        $i = 0;
        while (!feof($fh)) {
            $notes[$i] = fgets($fh);
            $i++;
        }
        fclose($fh);

        //移除首尾两个元素
        array_pop($notes);
        array_shift($notes);

        return $notes;
    }
```

```
        protected function getNote($noteIndex){
            $notes = $this -> getNotes();

            return $notes[$noteIndex];
        }

        //定义格式化输出所有日志的方法
        function displayNotes(){
            $notes = $this -> getNotes();

            //输出日志记录
            if(count($notes) > 0){
                foreach ($notes as $noteId => $note) {
                    $note = explode('|', $note);
                    echo  '<div class="note"><table>
                            <th class="center">
                                <td>#'.($noteId+1).'</td>
                                <td>'.$note[0].'</td>
                                <td>'.$note[1].'</td>
                                <td>'.$note[2].'</td>
                                <td>
                                    <a href="?modNote='.$noteId.'">修改</a> |
                                    <a href="?delNote='.$noteId.'">删除</a>
                                </td>
                            </th>
                            <tr class="indent">
                                <td colspan="4">'.$note[3].'</td>
                            </tr>
                        </table>
                    </div>';
                }
            } else {
                echo '<div class="note"><p class="center">暂无日志记录。
                </p></div>';
            }
        }

        //定义添加日志的方法
        function addNote($note){
            //将作者、日期和天气信息整合到日志中
            $note = implode('|', array($this -> getAuthor(),
                                        $this -> getWeather(),
                                        $this -> getNoteDate(),
                                        $note));

            //获取存储日志的文件名，并在其尾部追加新日志
            $filename = $this -> getFileName();
            $fh = fopen($filename, 'a+');
            if (fgets($fh) == '----start----'){
                fwrite($fh, $note);
            } else {
                fwrite($fh, $note."\r\n");
            }

            fclose($fh);
        }
    }
?>
```

在上面这段脚本中以“class notes”开始，定义了一个名为 notes 的类。紧接着定义了一系列属性，它们分别是写日志的时间（$noteDate）、日志的作者（$author）、写日志时的天气（$weather）以及用来存储日志的文本文件名（$filename）。在定义完 notes 类的属性后，在__construct 魔术方法中使用私有方法 setAuthor、setWeather、setNoteDate 和$setFileName 为这四个私有属性赋了值。这就意味着这四个私有属性的数据类型也就确定了，它们都是字符串型的变量。

在魔术方法中，还初始化了用来存储用户日志的文本文件。该文件的文件名是用户姓名加密后生成的一个字符串，扩展名为“.note”；若该文件不存在，则创建该文件。

接下来，我们定义了用来设置和获取私有属性的八个方法。设置私有属性的方法有 setAuthor、setWeather、setNoteDate 和 setFileName，这四个方法均为私有方法，要求提供一个参数且都没有返回值。类似这样的方法名以“set”打头的方法可以叫做“setter”方法。获取私有属性的方法有 getAuthor、getWeather、getNoteDate 和 getFileName，这四个方法中除了 getFileName 为受保护的方法外，其他三个方法为公有方法。这四个方法都没有参数要求且都返回对应私有属性的值。类似这样的方法名以“get”打头的方法叫做“getter”方法。

然后，我们定义了获取所有日志和指定日志的方法。它们分别是 getNotes 和 getNote，前者为公有方法，后者为受保护的方法；前者读取存储日志的文件，并将读取到的内容存入一个数组中，然后返回数组，该方法不要求提供任何参数；后者则引用了 getNotes 方法将日志文件中的内容存入到一个数组中，然后在数组用指定的索引获取对应的日志记录，该参数要求提供指定日志记录的索引值。另外，我们还定义了用于在页面上展示日志列表的 displayNotes 方法，该方法为公有方法，不要求提供参数，也不返回任何值，它只会按照既定的格式向页面输出日志记录。

最后我们定义了添加日志的方法 addNote，均为公有方法。添加日志的方法要求提供一个参数，且该方法不返回任何值，而是直接将操作完成后的日志记录重新写到文本文件中。

值得注意的是，我们在类中定义方法的时候，使用了$this 变量来访问类中定义的属性和方法。该变量代指类本身，而“->”符号则表示“访问”。以 setAuthor 方法为例：

```
private function setAuthor($author){
            $this -> author = $author;
        }
```

在定义 setAuthor 方法时，我们使用$this -> author 来访问 notes 类的$author 属性。由于$author 属性是 notes 类的私有属性，因此，该属性只能在类中通过$this 变量引用。同样地，如果需要在类中引入 setAuthor 方法，我们也可以通过$this 变量，如$this->setAuthor。这里，setAuthor 为 notes 类的私有变量，只能在类中通过$this 变量访问。

还有一点需要注意的是，在类中定义的方法，除了 setter 和 getter 类方法之外，其他的方法无论其状态如何，都不建议直接访问类的私有属性，而要通过 setter 和 getter 类方法来间接获取。这样，一旦修改了私有属性的名称，就只用修改 setter 和 getter 类中引用的相应的私有属性，否则，可能得在整个类的脚本中，修改散落在各处的私有属性名称。

现在通过实例化 notes 类来创建一个基于 notes 类的应用程序。

【例 12.4】 创建一个基于 notes 类的应用程序（\\Ex12-4.php）。

```
<?php
    include_once('class/class.notes.php');

    //定义 Notes 的使用者和当前天气情况
    $author = '张晓明';
    $weather = '晴';

    //初始化 notes 类
    $nts = new notes($author,$weather);


?>
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Notes</title>
        <meta charset="UTF-8">
        <link href="css/Ex12-04.css" rel="stylesheet">
    </head>
    <body>
        <div class="header">
            <h2>我的日志</h2>
            <hr />
        </div>
        <div class="notebox">
            <table>
                <tr>
                    <td class="triple indent">作者: <?php echo $nts->
                    getAuthor(); ?></td>
                    <td class="triple center">日期: <?php echo $nts->
                    getNoteDate(); ?></td>
                    <td class="triple center">天气: <?php echo $nts->
                    getWeather(); ?></td>
                </tr>
                <tr>
                    <td colspan="3">
                        <form class="center" action="?addNote" method="post">
                            <textarea rows="7" cols="100" name="note">请在
                            此输入日志。</textarea>
                            <input type="submit" value="我写完了">
                    </form>
                    </td>
                </tr>
            </table>
        </div>
        <div class="notelist" id="notelist">
        <?php
            //当用户添加新日志时执行如下脚本
            if(isset($_REQUEST['addNote'])){
                $note = $_REQUEST['note'];
                $nts -> addNote($note);

                $notes = $nts -> displayNotes();
            } else {
                $notes = $nts -> displayNotes();
            }
        ?>
```

```
        </div>
    </body>
</html>
```

上面这段脚本中，实例化了 notes 类，并在页面上引用了实例化后的 nts 对象的 getAuthor、getNoteDate 和 getWeather 方法获取了 nts 对象相关私有属性的值。

接着我们引用了 nts 对象的 getNotes 方法获取了存储在文本文件中的日志记录。若文本文件不存在，$nts 对象会自动创建该文件。

当用户提交表单后，我们引用了 nts 对象的 addNote 方法将用户提交的内容存储到指定的文本文件中。上面这段脚本最终运行的样子如图 12-1 所示。



图 12-1　创建一个基于 notes 类的应用程序

12.2.2　类的扩展和改写

基于对象的编程最令人称道的一点就是其扩展性和继承性。我们可以通过扩展一个类而获得一个新类，该新类继承了原有类的所有属性和方法，是原有类的子类。在上一小节中定义的类只有添加日志的功能，修改和删除的功能还没有实现。在本小节里，我们来扩展一下原有的 notes 类，创建一个 powerNotes 类，并为 powerNotes 类添加两个新方法 modNotes 和 delNotes。

【例 12.5】 创建一个扩展 notes 类的 powerNotes 类（\\class.powernotes.php）。

```
<?php
    //导入类文件
    include_once('class.notes.php');

    //创建一个扩展于 notes 类的新类，名为 powernotes
```

```
    class powerNotes extends notes {
        //为 powerNotes 类添加两个新的方法
        function modNote($noteIndex, $newNote){
            //获取指定记录
            $note = $this -> getNote($noteIndex);

            //替换现有记录中的日志内容
            $note = explode('|', $note);
            $note[3] = $newNote;

            //重新组合指定日志
            $note = implode('|', $note);

            //将指定日志还原到数组中
            $notes = $this -> getNotes();
            $notes[$noteIndex] = $note;


            //获取文件名
            $filename = $this -> getFileName();

            //删除之前的文件
            unlink($filename);

            //新建一个同名的文件
            $fh = fopen($filename, 'w');
            fwrite($fh, "----start----\r\n");
            fclose($fh);

            //遍历现有日志记录，并将其写入新建的同名文件中
            foreach ($notes as $key => $value) {
                if($key == $noteIndex){
                    $this -> addNote($value);
                } else {
                    $noteLength = mb_strlen($value, 'utf8');
                    $value = mb_substr($value, 0, $noteLength-2, 'utf8');
                    $this -> addNote($value);
                }

            }
        }

        function delNote($noteIndex){
            //获取日志记录
            $notes = $this -> getNotes();

            //删除指定的记录
            unset($notes[$noteIndex]);

            //获取文件名
            $filename = $this -> getFileName();

            //删除之前的文件
            unlink($filename);

            //新建一个同名的文件
            $fh = fopen($filename, 'w');
            fwrite($fh, "----start----\r\n");
            fclose($fh);
```

```
            //遍历现有日志记录，并将其写入新建的同名文件中
            foreach ($notes as $key => $value) {
                $noteLength = mb_strlen($value, 'utf8');
                $value = mb_substr($value, 0, $noteLength-2, 'utf8');
                $this -> addNote($value);
            }
        }
    }
?>
```

在定义 powerNotes 类时，我们使用了 extends 关键字（class powerNotes extends notes），表明现在定义的 powerNotes 类是 notes 类的子类，继承了 notes 类的所有公有和受保护的属性和方法。

需要注意的是，子类不能直接使用父类的私有属性和方法。

接着为 powerNotes 类定义了两个新的方法 modNotes 和 delNotes。这两个方法都是公有方法，前者需要提供修改的日志记录的索引和修改后的日志记录，后者要求提供需要删除的日志记录的索引。这两个方法都不会返回任何值。

编写这两个方法的思路是，从现有的文本文件中读取已有的日志记录，并将其存放到一个数组中。然后通过参数传递过来的日志记录索引在数组中查找相应的日志记录。在修改或删除指定的日志记录对应的数组元素后，删除现有文件，并创建一个同名的新文件。然后将修改或删除了指定日志记录对应的数组元素的数组逐条写入新文件中。再把修改数组写入到新文件的过程中，使用了继承自 notes 类的 addNotes 方法。

下面我们把 powerNotes 类实例化，创建一个名为$pn 的对象，然后引用 addNotes、modNotes 和 delNotes 方法执行创建、修改和删除日志的操作，看看会有什么结果。

【例 12.6】 创建、修改和删除日志的操作（\\Ex12-06.php）。

```
<?php
    include_once('class/class.powernotes.php');

    $author = "白秋明";
    $weather = "晴";

    $pn = new powerNotes($author, $weather);
?>
<!DOCTYPE html>
<html>
    <head>
        <title>添加、修改和删除日志记录</title>
        <meta charset="UTF-8" >
        <style>
            pre {
                font-size: 1.2em;
            }
        </style>
    </head>
    <body>
        <?php
            //在执行操作前输出现有记录
            echo '<pre>';var_dump($pn -> getNotes());echo '</pre>';
```

```
        ?>

        <form action="?addNote" method="post">
            <input type="submit" value="添加日志">
        </form>

        <form action="?modNote" method="post">
            <input type="submit" value="修改日志">
        </form>

        <form action="?delNote" method="post">
            <input type="submit" value="删除日志">
        </form>
    </body>
</html>

<?php
    if(isset($_REQUEST['addNote'])){
        $pn -> addNote("中秋节快乐!");

        //在执行操作后再次输出现有记录，看看是否添加了日志记录
        echo '<pre>';var_dump($pn -> getNotes());echo '</pre>';
    }

    if(isset($_REQUEST['modNote'])){
        $pn -> modNote(2, "国庆节快乐!");

        //在执行操作后再次输出现有记录，看看是否修改了指定的日志记录
        echo '<pre>';var_dump($pn -> getNotes());echo '</pre>';
    }

    if(isset($_REQUEST['delNote'])){
        $pn -> delNote(2);

        //在执行操作后再次输出现有记录，看看是否删除了指定的日志记录
        echo '<pre>';var_dump($pn -> getNotes());echo '</pre>';
    }
?>
```

在例 12.6 的脚本中，首先导入了 class.powernotes.php 文件，然后通过实例化 powerNotes 类创建了名为$pn 的实例。

接着，我们使用 powerNotes 类继承自 notes 类的 getNotes 方法输出了日志文件中的内容。

在该页面的 HTML 部分定义了三张表单，分别用来添加、修改和删除日志。添加的日志内容为“中秋节快乐”，修改后的日志内容为“国庆节快乐”。在执行了这三类操作中的任意一种后，系统都会再次输出日志文件的内容，供我们比对操作前后日志文件的变化。

需要注意的是，在修改和删除日志的两个方法中指定的日志记录索引为 2，因此要添加至少三条日志记录。

现在打开页面，连续单击三次“添加日志”按钮。页面的状态如下：

观察图 12-2，可以发现最后一次单击“添加日志”按钮前日志文件中有两条记录，而其后，日志记录增加到了三条。我们的 addNotes 方法奏效了。

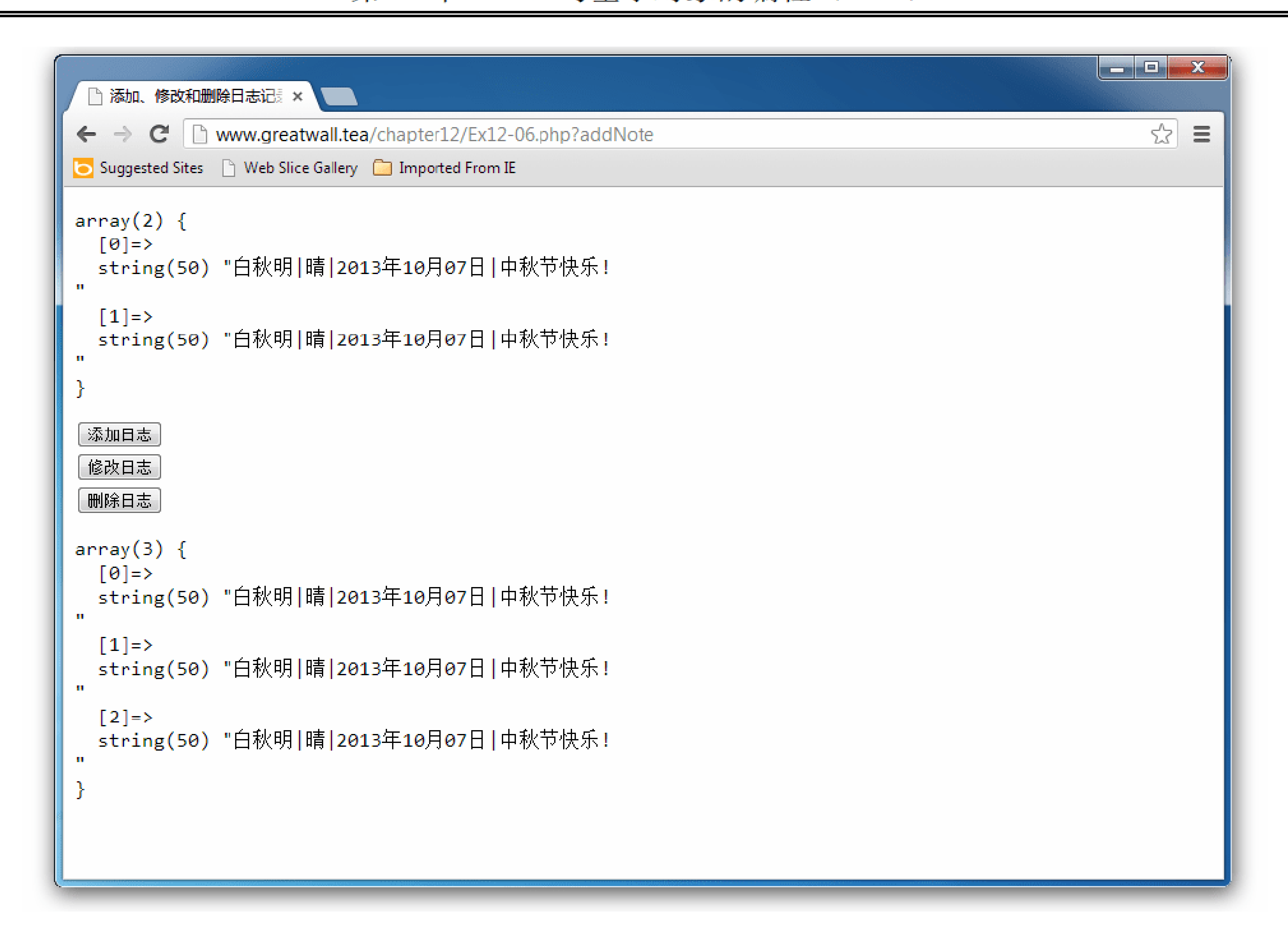


图 12-2　添加日志记录

现在我们再单击“修改日志”按钮，把索引号为 2 的日志记录的内容由“中秋节快乐”修改为“国庆节快乐”。修改后的页面如图 12-3 所示。

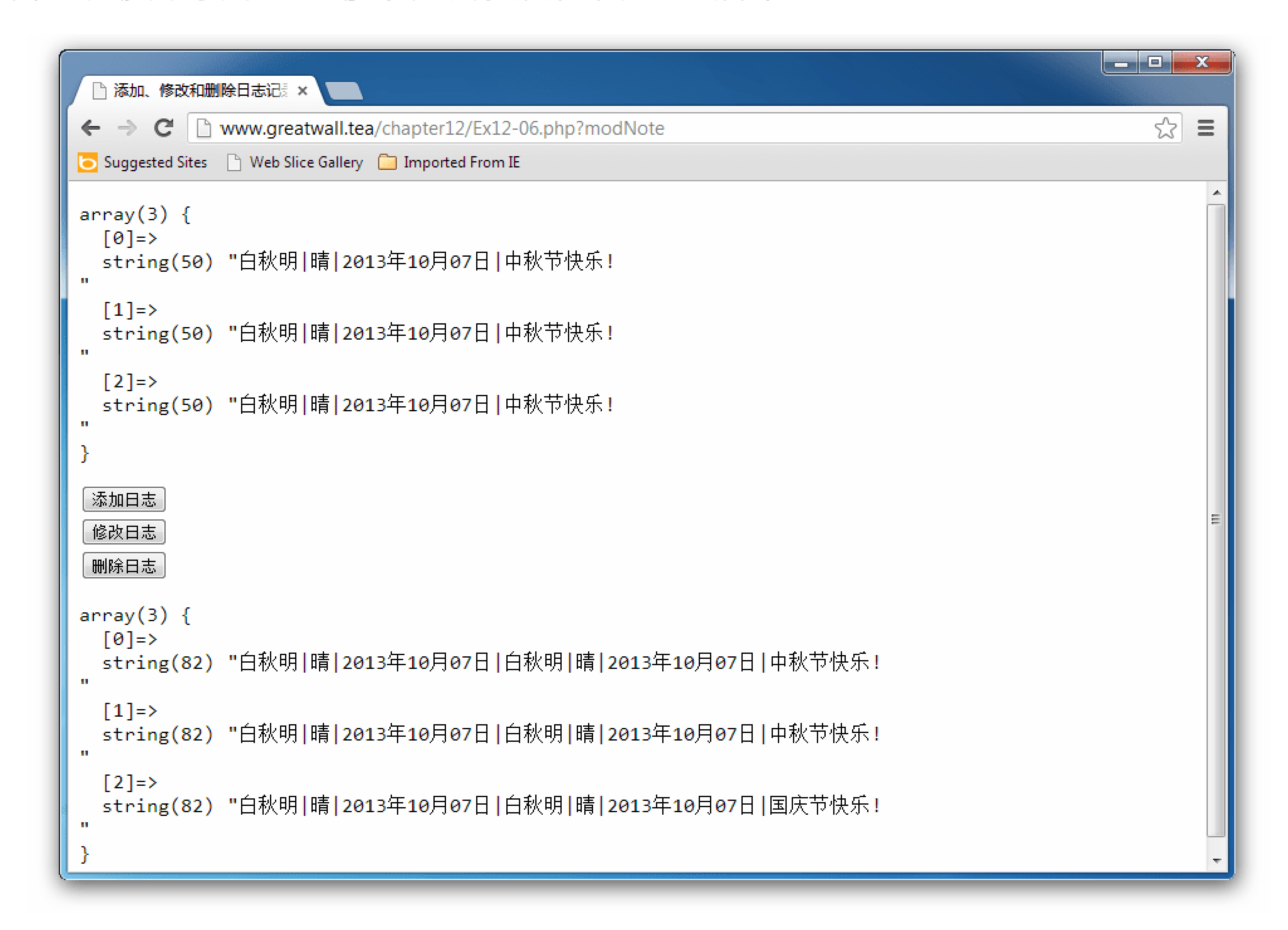


图 12-3　修改日志记录

在修改完成后，发现索引为 2 的日志记录的内容的确由“中秋节快乐”修改成了“国

庆节快乐”。但是为什么每条日志记录的内容变长了呢？

再单击“删除日志”按钮后，每条日志记录的内容变得更加长了。

这是什么原因造成的呢？原来，在 modNotes 和 delNotes 中都引用了 addNotes 方法，而 addNotes 方法中拼接日志记录的时候会把日志的作者、日志完成的时间和当时的天气情况连同添加的日志一起添加到日志记录中去。这在添加新记录的时候的确没有问题。但是在 modNotes 和 delNotes 中使用 addNotes 时，代入到 addNotes 中的参数不单单是日志记录的内容，而是包括了日志作者、日志完成的时间和当时的天气情况等信息的完整的日志记录。因而出现了如图 12-3 所示的情况。

修正这个 Bug 有两个思路，一个是修改在定义 modNotes 和 DelNotes 方法时代入到 addNotes 方法中的参数；另一个就是改写 addNotes 方法。

在基于对象的编程中，类间的继承和改写是常有的事。现在就在 powerNotes 类中，改写一下 addNotes 方法。

【例 12.7】 在 powerNotes 类中改写其继承自 notes 类的 addNotes 方法（\\class.powernotes.php）。

```
<?php
    //导入类文件
    include once('class.notes.php');

    //创建一个扩展于 notes 类的新类，名为 powernotes
    class powerNotes extends notes {
        //为 powerNotes 类添加两个新的方法
        function modNote($noteIndex, $newNote){
            …
            …
            //遍历现有日志记录，并将其写入新建的同名文件中
            foreach ($notes as $key => $value) {
                if($key == $noteIndex){
                    $this -> addNote($value, 'mod');
                } else {
                    $noteLength = mb_strlen($value, 'utf8');
                    $value = mb substr($value, 0, $noteLength-2, 'utf8');
                    $this -> addNote($value, 'mod');
                }
            }
        }

        function delNote($noteIndex){
            …
            …
            //遍历现有日志记录，并将其写入新建的同名文件中
            foreach ($notes as $key => $value) {
                $noteLength = mb_strlen($value, 'utf8');
                $value = mb_substr($value, 0, $noteLength-2, 'utf8');
                $this -> addNote($value, 'del');
            }
        }

        //改写继承自 notes 类的 addNotes 方法
        function addNote($note,$action = 'add'){
            //将作者、日期和天气信息整合到日志中
            if ($action == 'add') {
                $note = implode('|', array($this -> getAuthor(),
```

```
                                      $this -> getWeather(),
                                      $this -> getNoteDate(),
                                      $note));
            }

            //获取存储日志的文件名，并在其尾部追加新日志
            $filename = $this -> getFileName();
            $fh = fopen($filename, 'a+');
            if (fgets($fh) == '----start----'){
                fwrite($fh, $note);
            } else {
                fwrite($fh, $note."\r\n");
            }

            fclose($fh);
        }
    }
?>
```

所谓改写，其实就是把父类中的方法或属性再定义一遍。在上面这段脚本中，我们在继承自父类的 addNotes 方法中添加一个名为$action 的参数，并且这个参数是可选参数，如果不指定的话，那么这个参数就会使用默认值“add”。而只有当$action 参数值为“add”时，系统才会拼接日志内容和日志作者等信息。接下来，需要在 modNotes 和 delNotes 两个方法中引入 addNotes 时将$action 参数分别指定为“mod”和“del”（其实这里的值是什么并不重要，只要不是“add”就好）。

现在再来运行一下例 12.6 中的脚本。添加三条记录，然后修改和删除第三条记录。你会发现，刚才发现的 Bug 已经成功地清除掉了。

用改写类的方法是不是要比修改在定义 modNotes 和 DelNotes 方法时代入到 addNotes 方法中的参数容易的多呢？

12.2.3　修饰词

知道如何改写继承自父类的方法和属性后，我们就可以对所有继承自父类公有的和受保护的方法与属性进行改写。不过对于那些可以在子类中使用，却不建议在子类中改写的方法和属性，我们就得在父类中定义这些方法和属性时使用 final 修饰词。

使用 PHP 基于对象的编程中经常用到的修饰词有如下几种。

（1）public

使用 public 修饰的方法和属性称为公有方法和属性，可以在定义该方法和属性的类的子类和实例中使用。默认情况下，public 修饰词不是必需的，所有不带修饰词的方法和属性都是公有的。

（2）private

使用 private 修饰的方法和属性称为私有方法和属性，只能在定义该方法和属性的类中使用。在其子类和实例中，这些方法和属性都是不可见的。默认情况下，只有在定义某个方法或属性时使用了 private 修饰词，该方法或属性才是私有的。

（3）protected

使用 protected 修饰的方法和属性称为受保护的方法和属性，只能在定义该方法和属性

的类及其子类中使用。对于定义该方法和属性的类及其子类的实例来说，该方法和属性均不可见。默认情况下，只有在定义某个方法或属性时使用了 protected 修饰词，该方法或属性才是受保护的。

（4）static

使用 static 修饰的方法和属性称为静态方法和静态属性，它在基于对象的编程中至关重要。对于在某个类中定义的静态方法和静态属性，我们可以在该类没有被实例化的情况下直接调用这些方法和属性。一个静态方法和属性，其状态对于定义该方法和属性的类及该类的实例来说都是一致的。举个例子，当改变了某静态属性的值，那么基于定义这个静态属性的类的所有实例都会使用该属性改变后的值来代入运算。

（5）final

使用 final 修饰的方法和属性称为死方法和死属性，这些方法或属性无论在何时都是不可以被改写的。在 final 修饰词前还可以添加上述三类修饰词，共同决定它们修饰的方法或属性的适用范围。比如，如果我们使用 private final 来修饰一个方法，那么这个方法只能在定义它的类中使用，且不可被改写；如果我们使用 protected final 来修饰一个方法，那么这个方法只能在定义它的类及其子类中使用，且在该类及其子类中均不可被改写。

另外，与上述三种修饰词不同的是，final 修饰词还可以修饰类名。如果在定义某个类时使用了 final 修饰词，那么就不能创建基于该类的子类。

（6）interface

使用 interface 修饰的类称为接口类。在接口类中，只需声明方法和属性，不需要为这些方法和属性编码和赋值。当我们使用 implements 关键字来创建一个基于某接口类的类时，在新创建的类中，必须定义那些已经在接口类中声明的方法和属性。

比如应用程序允许使用 MySQL、PostgreSQL 和 SQLite 三种类型的数据库来存储数据。为了应用程序在对接这三类数据库时使用相同的方法和属性，我们可以定义一个接口类，然后在这个接口类中声明需要的方法和属性。接下来，就可以使用 implements 关键字创建三个类，分别针对不同类型的数据库来实现这些方法和属性。

（7）abstract

使用 abstract 修饰的类称为抽象类。所谓抽象类，顾名思义，就是把若干类中公用的部分抽离出来，放到一个抽象类中以减少脚本的冗余程度。抽象类的作用与接口类类似。不同的是，抽象类中除了声明方法和属性之外，还可以为其添加功能或赋值。另外，需要注意的是，在使用抽象类时，使用的关键字是 extends，而不是 implements。

在这些类修饰词中，前四种只适用于在类中修饰方法名和属性名，第五种则同时适用于修饰类名和类中的方法名与属性名，而最后两种则只适用于修饰类名。

12.2.4　一些魔术方法

在第 8 章学习了两个魔术方法_ _construct()和_ _destruct()。知道了魔术方法的名称前会有两道下画线（“_ _”）。本小节将了解到另外三个魔术方法_ _get()、_ _set()和_ _isset()。

在编写 notes 类的时候，大家写了八个类用 set 和 get 开头的方法用来设置和获取存储日志的文本文件名称、日志的作者、写作日期和当时的天气情况。在如此短小精悍的小程序里，就要写八个 setter 和 getter 方法。那要是应用程序的功能更复杂，要写的 setter 和 getter

可就太多了。为了避免出现这种繁琐的情况，可以使用_ _set()和_ _get()魔术方法。

使用_ _set()魔术方法设置了许多专属于某个对象的临时属性之后，我们可能一时没有办法弄清楚是不是设置了某个属性。这时，就要通过_ _isset()方法来进行判断，以免出现使用没有设置的属性的情况。

下面来看一个简单的使用_ _set()、_ _get()和_ _isset()魔术方法的例子。

【例 12.8】 使用_ _set()、_ _get()和_ _isset()魔术方法的例子（\\class.character.php）。

```
<?php
    class character {
        //设置一个私有数组变量
        private $properties = array();

        //定义一个  get()魔术方法
        function   get($property){
            return $this -> properties[$property];
        }

        //定义一个__set()魔术方法
        function   set($property, $value){
            $this -> properties[$property] = $value;
        }

        //定义一个  isset()魔术方法
        function __isset($property){
            return isset($this -> properties[$property]);
        }
    }
?>
```

在这个名为 character 的类中，我们定义了一个私有属性，其数据类型为数组，然后使用_ _et、_ _get 和_ _isset 定义了三个公有方法。其中，_ _set 方法有两个参数，没有返回值；_ _get 和_ _isset 方法均有一个参数，前者返回获取到的值，后者返回指定参数是否存在的判断结果；_ _unset 方法有一个参数，且没有返回值。

在定义好这个类之后，我们再实例化这个类，来见证一下魔法的神奇。

【例 12.9】 创建一个基于 character 的类并使用魔术方法。

```
<?php
    include once('class/class.character.php');

    //创建一个基于 character 类的对象，将其命名为$chara[1]
    $chara[1] = new character();

    //现在我们定义一下$chara1 的一些基本信息
    $chara[1] -> firstName = 'John';
    $chara[1] -> lastName = 'Dow';
    $chara[1] -> gender = 'Male';
    $chara[1] -> occupation = 'Product Manager';
    $chara[1] -> company = 'Parachute Inc.';

    //再创建一个基于 character 类的对象，将其命令为$chara[2]
    $chara[2] = new character();

    //现在我们定义一下$chara2 的一些基本信息
    $chara[2] -> firstName = 'Jane';
```

```
    $chara[2] -> lastName = 'Dow';
    $chara[2] -> gender = 'Female';
    $chara[2] -> occupation = 'PR Coordinator';
    $chara[2] -> company = 'Parachute Inc.';
    $chara[2] -> undergrad = 'Cornell University (2007-2011)';
    $chara[2] -> grad = 'Stanford University (2011-2013)';

    //再创建第三个基于 character 类的对象，将其命名为 chara[3]
    $chara[3] = new character();

    //现在我们定义一下$chara3 的一些基本信息
    $chara[3] -> firstName = 'Max';
    $chara[3] -> lastName = 'Poll';
    $chara[3] -> gender = 'Male';
    $chara[3] -> occupation = 'Sales Director';
    $chara[3] -> company = 'Parachute Inc.';
    $chara[3] -> undergrad = 'Carnegie Mellon University (2003-2007)';
?>
<!DOCTYPE html>
<html>
    <head>
        <title>Name Box</title>
        <meta charset="UTF-8">
        <link href="css/Ex12-09.css" rel="stylesheet">
    </head>
    <body>
        <div class="container">
        <?php
            //接着我们输出一下这三个人的信息
            foreach ($chara as $key => $value) {
                echo '<div class = "float-box"><p class="num">#'.$key.'
                </p><p class="name">'.
                ($value ->__isset('firstName')?$value -> firstName : N/A').' '.
                ($value ->   isset('lastName') ? $value -> lastName :
                'N/A').' <span class="title">'.
                ($value -> __isset('gender') ?
                (($value -> gender == 'Male') ? 'Mr.' : 'Mrs./Miss') :
                'N/A').'</span></p><p class="pc">'.
                ($value -> __isset('occupation') ? $value -> occupation :
                'N/A').'</p><p class="pc">'.
                ($value -> __isset('company') ? $value -> company :
                'N/A').'</p><ul class="edu"><li>'.
                ($value ->   isset('undergrad') ? $value -> undergrad :
                'N/A').'</li><li>'.
                ($value ->   isset('grad') ? $value -> grad : 'N/A').'
                </li></div>';
            }
        ?>
        </div>
    </body>
</html>
```

例 12.9 创建了三个基于 character 类的实例，分别为 chara1、chara2 和 chara3。然后我们访问并设置了这三个对象中并不存在的私有变量，如 firstName、lastName 和 gender 等。接着在页面的 HTML 部分，输出了刚才设置的这些私有变量。

神奇的事情出现了：页面没有报错，并且输出了对应的值。

在实例化这个类的页面上，我们在输出某个属性之前使用__isset()判断该属性是否存在。若存在，则输出该属性的值；若不存在，则输出“N/A”。就像下面这句脚本一样：

```
($value -> __isset('firstName') ? $value -> firstName : 'N/A')
```

页面中的所有元素都是通过这种方式输出到页面中的。

这个页面最终的效果如图 12-4 所示。

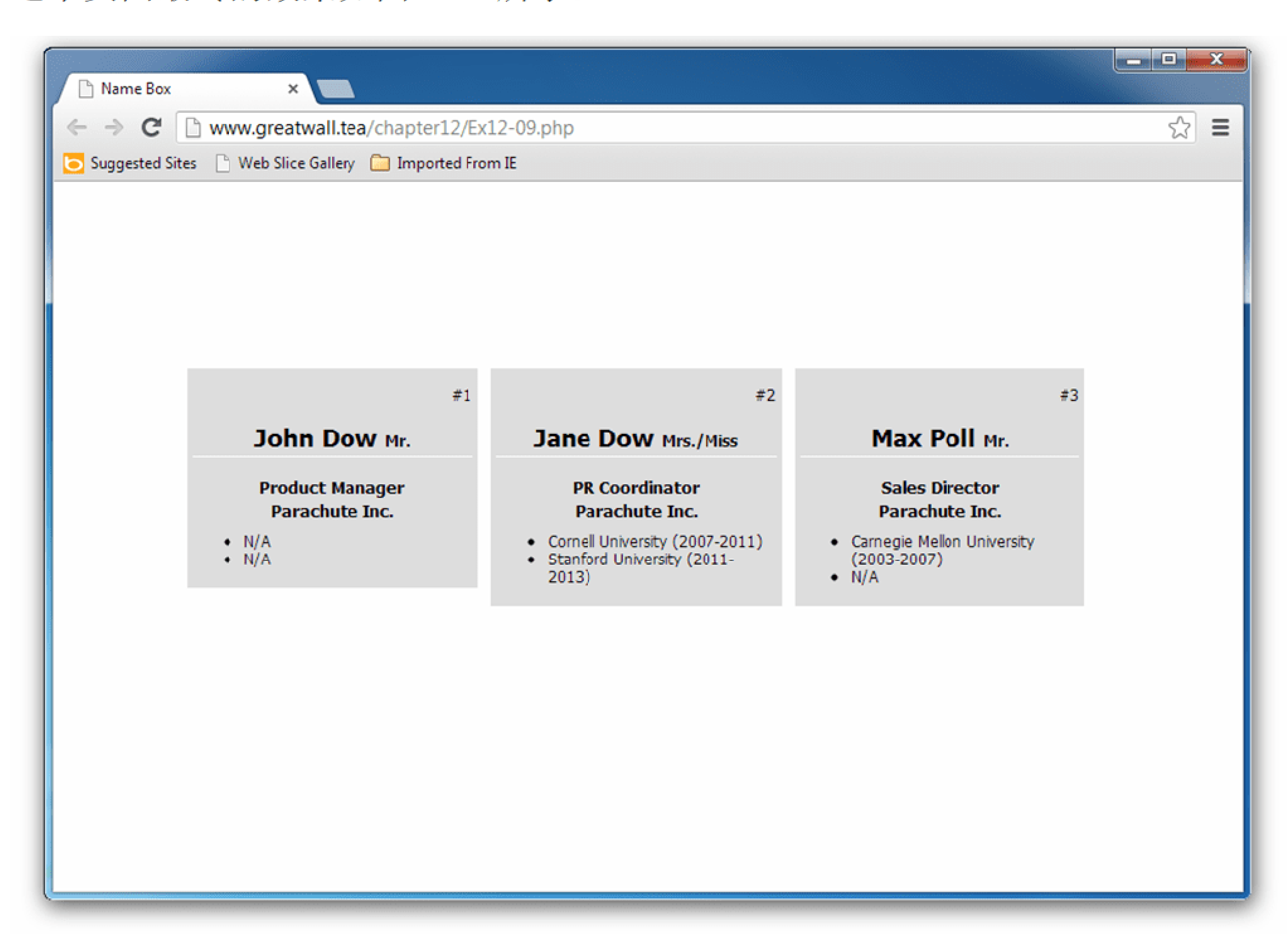


图 12-4　魔术方法的使用

看到这里，大家应该明白了如何使用这些魔术方法了。不过需要再提醒的是，合理使用魔术方法才是正道。虽然魔术方法可以方便为对象设置临时属性，但是过多地依赖魔术方法可能会导致参数过于复杂，从而降低脚本的可读性和可维护性。

所以，在使用这些魔术方法时，万望三思。

12.3　进阶 OOP

在上一节里，我们掌握了基于对象编程的基础知识。在本节里，我们将基于对象编程的一些高级功能进行讲解。

12.3.1　摸清类的情况

在有些时候，我们需要对类的情况进行摸底，以便更好地使用类及类中定义的方法和属性。因此，PHP 为我们提供了一些函数来获取指定类的信息，从而摸清类的情况。

如果实例化一个不存在的类，系统会报错。为了避免出现这种情况，我们可以使用 class_exists()函数来判断一个类是否存在。这样一来，我们就可以保证要实例化的类是一定

存在的了。

【例 12.10】 使用 class_exists()函数判断类是否存在。

```
<?php
    include_once('class/class.calc.php');

    if(class_exists('calc')){
        $calc = new calc();
        echo $calc -> getVarA().' + '.$calc -> getVarB().' = '.$calc -> add();
    } else {
        die ('不存在名为"calc"的类。');
    }
?>
```

在例 12.10 中，使用 class_exists()判断类 calc 是否存在。若存在，则创建一个基于该类的名为 calc 的对象，然后通过对象调用 getVarA、getVarB 和 add 方法输出算式“5 + 6 = 11”；若 calc 类不存在，则输出“不存在名为 calc 的类”。

如果想看到类不存在时脚本的运行结果，可以把 include_once 一句给注释掉。

有的时候，我们想知道现在可供使用的类有哪些，以便在其中选择一些类。PHP 提供了一个名为 get_declared_classes 方法，可以帮助我们做到这一点。在例 12.11 的例程中，我们导入了类文件 class.calc.php，然后使用 var_dump 方法输出了 get_declared_classes()函数的运行结果。你会发现 get_declared_classes()函数返回的是一个数组，数组中全是字符串类型的元素。每个字符串元素都代表着一个已经被声明的类。滚动鼠标到页面的最底端，可以找到导入的 calc 类。

【例 12.11】 使用 get_declared_classes 方法获取当前已声明的所有类。

```
<?php
    include_once('class/class.calc.php');

    echo '<pre>';var_dump(get_declared_classes());echo '</pre>';
?>
```

有时候，我们在一个页面中会导入多个类文件，可能还会创建多个基于同一个类的对象。创建的对象一多，就会让人犯迷糊。PHP 提供了一个名为 is_a()的函数，可以帮助我们判断某个对象到底是不是基于某个类的。这个函数的参数是某个对象的名称和某个类的名称。

【例 12.12】 使用 is_a()函数判断某对象是否为某类的实例。

```
<?php
    include_once('class/class.calc.php');

    //创建一个基于 calc 类的实例
    $calc = new calc();

    //判断对象$calc 是否是一个基于 calc 类的实例
    echo (is_a($calc,'calc'))?'对象$calc 是一个基于 calc 类的实例。':'对象$calc
    不是一个基于 calc 类的实例。';
?>
```

使用 is_a()函数只能判断某个对象是不是某个已知类的实例。如果想要知道某个对象到底是哪个类的实例，它就无能为力了。不过 PHP 又准备了一个名为 get_class()的函数，

可以帮助我们做到这一点。这个函数只有一个参数，返回一个字符串类型的值。

【例 12.13】 使用 get_class()函数查找某对象所属类。

```
<?php
    class parentClass {

        //定义一个公有属性用于获取基于该类的对象所属的类
        function getClass() {
            echo get_class();
            echo '<br>';
            echo get_class($this);
        }
    }

    //创建一个继承 parentClass 类的名为 childClass 的类
    class childClass extends parentClass {

    }

    //创建一个基于 childClass 类的名为$cc 的对象
    $cc = new childClass();

    //直接调用 get_class 函数输出对象$cc 所属的类
    echo get_class($cc);
    echo '<br>';

    //调用对象$cc 的公有方法 getClass
    $cc -> getClass();
?>
```

例 12.13 定义了两个类，分别为 parentClass 和 childClass，其中 childClass 继承自 parentClass。在 parentClass 中，我们定义了一个公有方法 getClass，并在当中使用 echo 输出 get_class()和 get_class($this)的值。

接着我们创建了一个基于 childClass 类的对象，名为$cc，然后使用 echo 输出了对象$cc 所属的类。最后调用了对象$cc 的 getClass 方法。

这段脚本运行的结果如下：

```
childClass
parentClass
childClass
```

第一行输出的“childClass”是在对象$cc 被创建后直接输出的 get_class($cc)的值。第二行和第三行输出的是通过$cc 对象调用的继承自 parentClass 的 getClass 方法的运行结果。其中，第二行为 getClass 方法中的“echo get_class()”输出的结果，而第三行则是 getClass 方法中的“echo get_class($this) ”输出的结果。

通过这段例程，我们可以知道，如果使用不带参数的 get_class()函数，返回的是由其运行范畴决定的。在上面这段例程中，不带参数的 get_class()就运行在 parentClass 的范畴里，所以返回的值为“parentClass”。如果使用带参数的 get_class($this)，那么，当我们实例化 parentClass 类时，$this 指代的就是基于 parentClass 类的实例；当我们实例化 childClass 类时，$this 指代的就是基于 childClass 类的实例。在上面这段例程中，childClass 是继承自 parentClass 的子类，而对象$cc 又是 childClass 的实例化对象，所以这时$this 指代的应该是

childClass，那么它返回“childClass”就无可厚非了。

12.3.2　迭代器

在第 6 章中，我们学习过如何遍历数组。我们也可以用相同的方法来遍历对象中的公有属性和方法，而这种方法是 PHP 4 时代就有的方法。在 PHP 5 时代，PHP 为我们准备了迭代器接口类，通过使用迭代器来遍历数组和对象。

所谓迭代器，就是用来遍历数组和对象的类。它们通常使用 implements 关键字定义，拥有统一的方法。PHP 中提供的迭代器有如下两种。

- Iterator：Iterator 称为简单迭代器。它是一个预定义的接口类，定义了 current、key、next、rewind 和 valid 五种必选方法接口。通过自定义这些方法接口的实现方式，因地制宜地实现对数组和对象的遍历。
- IteratorAggregate：IteratorAggregate 称为聚合迭代器。它只有一个名为 getIterator 的预定义的方法接口。通过在该方法接口中引用外部迭代器，可以实现对数组和对象的遍历。

下面，我们就来创建一个简单迭代器和一个聚合迭代器。前者用来读取日志文件目录下的所有日志文件中的日志记录；后者则用来读取日志文件目录下的所有日志文件。

【例 12.14】 简单迭代器 notesIterator（\\class.notesInterator）。

```
<?php

    //定义一个用于遍历使用 notes 类创建的所有日志的迭代器
    class notesIterator implements Iterator {
        //定义一些私有变量
        private $files = array();
        private $notes = array();
        private $position;

        //定义一些常量
        const FILE_ROOT = 'file/';

        //初始化迭代器
        function __construct(){
            //获取指定目录下的所有日志文件
            $dh = opendir(self::FILE_ROOT);
            $i = 0;
            while ($fn = readdir($dh)) {
                if(substr($fn, -6) == '.notes' &&
                   strlen($fn) > 2) {
                     $this -> files[$i] = $fn;
                     $i++;
                }
            }

            //获取所有日志文件中存储的日志
            $i = 0;
            foreach ($this -> files as $fnid => $fn) {
                $fh = fopen(self::FILE_ROOT.$fn, 'r');
                while($line = fgets($fh)){
```

```
                        if($line <> "----start----\r\n"){
                            $this -> notes[$i] = mb_substr($line,0,mb_strlen
                            ($line, 'utf-8')-2,'utf-8');
                            $i++;
                        }
                    }

                    fclose($fh);
                }

                //初始化迭代器指针
                $this -> position = 0;
            }

            function rewind(){
                $this -> position = 0;
            }

            function current(){
                return $this -> notes[$this -> position];
            }

            function next(){
                ++$this -> position;
            }

            function key(){
                return $this -> position;
            }

            function valid(){
                return isset($this -> notes[$this -> position]);
            }
        }
?>
```

在 notesIterator 迭代器中，我们先封装了三个私有变量 files、notes 和 position。其中，files 和 notes 的数据类型为数组，而 position 数据类型未定。接着我们封装了一个常量 FILE_ROOT，其值为“file/”。表明存储日志文件的目录名称。

在__construct()魔术方法中，我们通过 opendir()、readdir()、fopen()和 fgets()函数将“file/”目录下的日志文件和这些日志文件里的日志记录读取到了 files 和 notes 两个私有属性中。

随后，我们定义了 iterator 接口类预定义的五个方法接口 rewind、current、next、key 和 valid 的实现方式，这个迭代器就写好了。

现在再来看看如何定义聚合迭代器。通过引入外部迭代器，我们定义迭代器的过程会变得更加简单。

【例 12.15-1】 聚合迭代器 filesIterator（\\class.fileIterator.php）。

```
<?php
    class filesIterator implements IteratorAggregate {
        //定义一些私有变量
        private $files = array();

        //定义一些常量
```

```
        const FILE_ROOT = 'file/';

        function __construct() {
            //获取指定目录下的所有日志文件
            $dh = opendir(self::FILE_ROOT);
            $i = 0;
            while ($fn = readdir($dh)) {
                if(substr($fn, -6) == '.notes' &&
                   strlen($fn) > 2) {
                    $this -> files[$i] = $fn;
                    $i++;
                }
            }
        }

        function getIterator(){
            return new ArrayIterator($this->files);
        }
    }
?>
```

在 fileIterator 类中，我们先封装了一个私有变量$file，其数据类型为数组。接着封装了一个常量 FILE_ROOT，其值为“file/”，表明存储日志文件的目录。然后，我们使用 opendir()和 readdir()函数将指定目录中的所有后缀名为“.notes”的日志文件的文件名存储到私有属性$files 数组中。

最后，我们实现了 IteratorAggregate 类预定义的 getIterator 方法接口，引入了一个外部迭代器 ArrayIterator 用来遍历 filesInterator 类的私有属性 files。

现在来创建基于 notesIterator 类和 filesIterator 类的实例，将其命名为$ni 和$fi。

【例 12.15-2】 创建一个基于 notesIterator 类的名为$ni 的实例（\\Ex12-15.php）。

```
<?php
    include_once('class/class.notesiterator.php');
    include_once('class/class.filesiterator.php');

    //使用 notesIterator 迭代器遍历所有日志文件中的日志记录
    $ni = new notesIterator();

    foreach ($ni as $key => $value) {
        echo '<pre>';var_dump($key,$value);echo '</pre>';
    }

    //使用 filesIterator 迭代器遍历存储日志文件的目录中所有日志文件
    $fi = new filesIterator();

    foreach ($fi as $key => $value) {
        echo '<pre>';var_dump($key,$value);echo '</pre>';
    }
?>
```

在上面这段脚本中，我们实例化了 notesIterator 类和 filesnotes 类，并将实例化后的对象分别赋值给变量$ni 和$fi。然后使用 foreach 语句输出了这两个对象的内容。

例 12.15-2 中的脚本运行结果如图 12-5 所示。

通过图 12-5 可以发现$ni 和$fi 对象分别把私有属性$notes 和$files 里的内容全部读取出来了，我们不需要定义任何 getter 方法就可以获取这两个私有属性中的所有内容。是不是很方便呢？

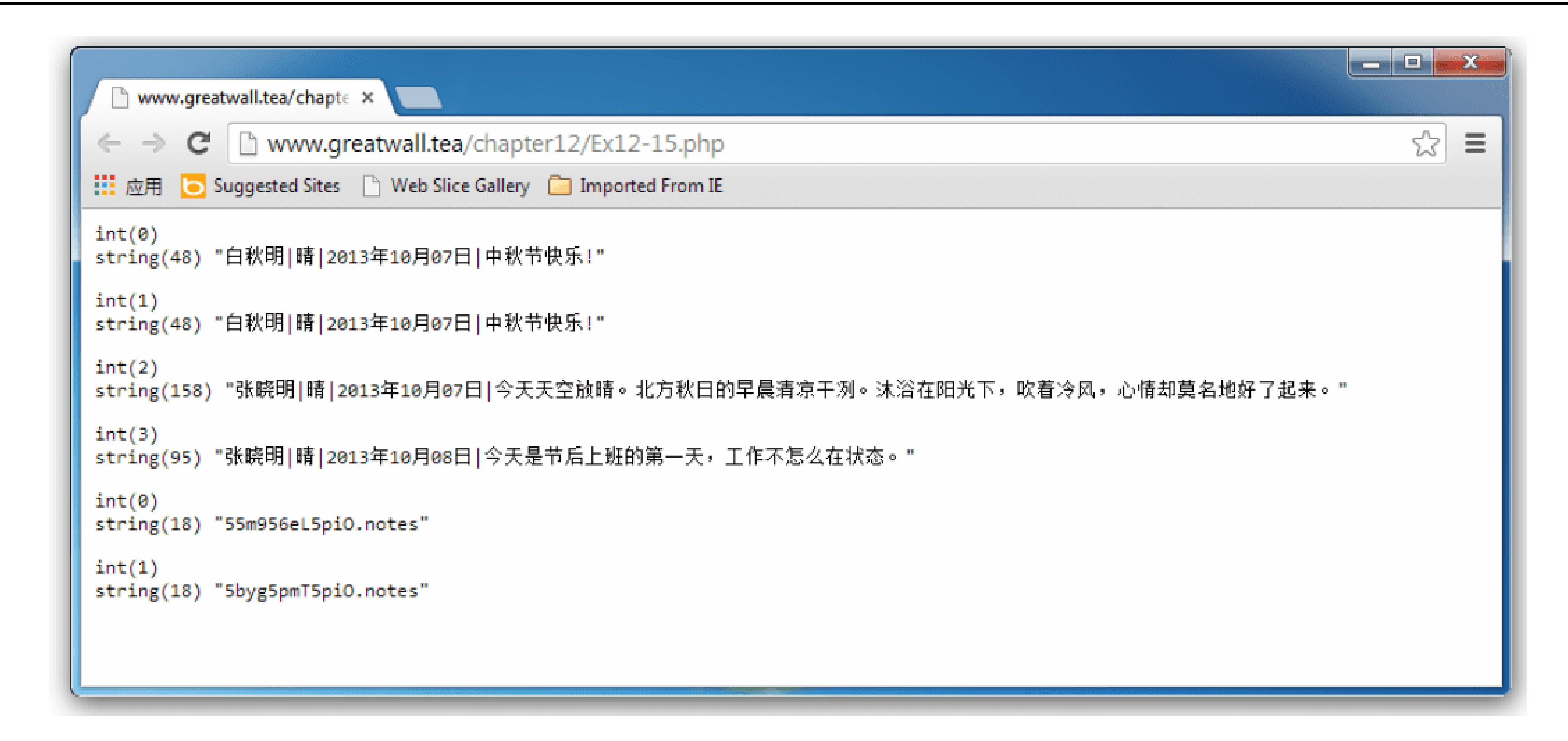


图 12-5　notesIterator 实例化后的输出结果

12.3.3　数组对象

在 PHP 4 时代，数组和对象是不同的。但是到了 PHP 5 时代，除了有数组和对象之外，还有数组对象。通过数组对象，我们可以很方便地使用基于对象的方法访问和修改数组。下面来看看这个奇妙的数组对象。

在 PHP 5 中，PHP 预定义了一个 ArrayObject 类。这个类有如下几个常用的方法。

- append：该方法是用来向数组添加元素的。
- getIterator：该方法会创建一个简单迭代器对象，并返回该对象。这样我们就可以使用 Iterator 对象预定义的方法来遍历数组。
- offsetExists：该方法用来判断某个指定的索引是否已经定义。
- offsetGet：该方法用来获取某个指定的索引对应的值。
- offsetSet：该方法用来设置某个指定的索引对应的值。
- offsetUnset：该方法用来取消对某个指定索引的定义。

现在我们来定义一个数组对象。

【例 12.16】 创建数组对象（\\Ex12-16.php）。

```
<?php
    //定义一个数组
    $users = new ArrayObject(array("hasin"=>"hasin@pageflakes.com",
                          "afif"=>"mayflower@phpxperts.net",
                          "ayesha"=>"florence@pageflakes.net"));


    //定义一个迭代器
    $iterator = $users->getIterator();

    //使用迭代器遍历数组
    while ($iterator->valid())
    {
        echo "{$iterator->key()}'s Email address is
              {$iterator->current()}<br>";
```

```
        $iterator->next();
    }
?>
```

在上面这段脚本中，首先创建了一个基于 ArrayObject 类的对象，将其命令为$user，用来存储用户相关信息。接着使用$user 对象的 getIterator 方法创建了一个迭代器；最后遍历这个迭代器中的所有信息。

12.3.4　对象序列化

对象序列化是指把已创建的基于某个类的对象转换成可在页面间传递的字节流，这也是对象序列化诞生的最重要的一个原因。它包括两个步骤：序列化和反序列化。前者通过 serialize()函数序列化一个对象，使其转化成一个二进制字符串；后者则使用 unserialize()函数将这个二进制字符串转换成之前被序列化的对象。

通过这样的方法在页面间传递的对象，其数据和内存结构都是完整的。不过需要注意的是，序列化函数 serialize 只能序列化对象的属性，不能序列化对象的方法，也就是说在需要反序列化由某个对象转化而来的二进制字符串时，需要导入派生该对象的类。

我们看一个对象序列化的例子。

【例 12.17】　对象的序列化（\\Ex12-17.php）。

```
<?php
    //开启 session
    session_start();

    //导入 calc 类所在的类文件
    include_once('class/class.calc.php');

    //创建一个基于 calc 类的对象，将其命名为$c
    $c = new calc();

    //序列化该实例，并将序列化后的字节流存入名为$cString 的 session 变量中
    $_SESSION['cString'] = serialize($c);

    //创建一个链接，用来转向另一个页面
    echo '<a href="Ex12-18.php">开始传递对象</a>';
?>
```

在例 12.17 中开启了 session 功能，打算使用 session 变量在两个页面间传递由 calc 类派生出的对象。接着我们导入了 calc 类，创建了一个基于 calc 类的对象，并将其命名为$c。然后使用 serialize()函数将对象$c 序列化，并将序列化后的二进制字符串存入 session 变量$_SESSION['cString']中。最后，在页面上输出了一个链接，链向对象传递的目的页面。

下面，我们来编写对象传递的目的页面。

【例 12.18】　对象的反序列化（\\Ex12-18.php）。

```
<?php
    //开启 session
    session_start();

    //导入派生对象$c 的类
    include_once('class/class.calc.php');
```

```
    //判断 session 变量 cString 是否已设置，若已经设置则反序列化该 sesssion 变量
    if(isset($_SESSION['cString'])){
        $c = unserialize($_SESSION['cString']);

        //使用对象$c 的 getVarA()、getVarB()和 add 方法输出"5 + 6 = 11"
        echo $c -> getVarA().' + '.$c -> getVarB().' = '.$c -> add();
    } else {

        //若不存在名为 cString 的 session 变量，则输出如下内容
        echo '对象未定义。';
    }
?>
```

在这个页面中，我们也开启了 session 功能以便接受由其他传递过来的 session 变量。由于 serialize()函数只能传递对象的属性，而不能传递方法，所以我们在对象传递的目的页面上也导入了派生对象$c 的 calc 类。接着判断 session 变量是否已设置：若未设置，则输出“对象未设置”；若已设置，则使用 unserialize()函数反序列化存储在 session 变量中的二进制字节流，并将反序列化后的对象存储在同名对象$c 中。最后，我们使用对象$c 的 getVarA、getVarB 和 add 方法输出了“5 + 6 = 11”这个算式。

通过上面两个例程，我们知道了如何通过序列化对象，从而方便地在页面间传递对象。这里由于篇幅的原因，使用了最简便的 session 变量。试问，如果不使用 session 变量，如何实现类似例 12.17 和例 12.18 之间的对象传递呢？大家可以思考一下。

12.3.5　对象的克隆

为了讲清楚对象的克隆，我们先来看一段脚本。

【例 12.19】 对象的克隆（\\Ex12-19.php）。

```
<?php
    include_once('class/class.calc.php');

    $sample1 = new calc();

    //使用对象$sample1 的 setter 方法设置私有属性 a,b 的值分别为 10,20
    $sample1->setVarA(10);
    $sample1->setVarB(20);
    //输出对象$sample1 的 add 方法的执行结果
    echo $sample1->getVarA().' + '.$sample1->getVarB().' = '.$sample1->add().
    '<br>';

    //将对象$sample1 赋值给对象$sample2
    $sample2 = $sample1;

    //使用对象$sample2 的 setter 方法设置私有属性 a,b 的值分别为 8,9
    $sample2->setVarA(8);
    $sample2->setVarB(9);

    //输出对象$sample1 的 add 方法的执行结果
    echo $sample1->getVarA().' + '.$sample1->getVarB().' = '.$sample1->add().
    '<br>';

    //将对象$sample2 克隆给对象$sample3
```

```
    $sample3 = clone $sample1;

    //使用对象$sample3 的 setter 方法设置私有属性 a,b 的值分别为 10，20
    $sample3->setVarA(10);
    $sample3->setVarB(20);

    //再次输出对象$sample1 的 add 方法的执行结果
    echo $sample1->getVarA().' + '.$sample1->getVarB().' = '.$sample1->add();
?>
```

在上面的脚本中，首先创建了一个基于 calc 类的对象，将其命名为 sample1。接着使用 sample1 对象的 setter 方法将其私有属性 a 和 b 的值分别设置为 10 和 20；将对象 sample1 赋值给变量 sample2。然后使用 sample2 的 setter 方法将其私有属性 a 和 b 的值分别设置为 8 和 9。最后把对象 sample2 克隆给 sample3，并通过其 setter 方法将其私有属性 a 和 b 分别设置为 10 和 20。

在使用 setVarA 和 setVarB 方法修改了 sample1 对象的私有属性后，我们使用对象 sample1 的 getVarA、getVarB 和 add 方法输出了一个算式。在把 sample1 对象赋值给 sample2 后，使用对象 sample1 的 getVarA、getVarB 和 add 方法又输出了一个算式。在克隆后，我们又一次使用了对象 sample1 的相同方法输出了一个算式。

大家可以猜一下，这三次输出的结果是一样的吗？结果应该是“10 + 20 = 30”，还是“8 + 9 = 17”呢？

首先，第一次输出的结果为“10 + 20 = 30”，因为我们使用两个 setter 方法修改私有属性 a 和 b 的值。第二次输出的结果为“8 + 9 = 17”，因为在使用 sample2 对象的两个 setter 方法修改了其私有属性的同时，sample1 对象的私有属性也被修改了。而第三次输出的结果依然为“8 + 9 = 17”，因为 sample3 对象和 sample1 对象是两个相互独立、互不影响的两个对象，使用 sample3 对象的 setter 方法修改其私有属性与 sample1 的私有属性没有什么关系。

如果大家还想不通的话，可以把对象$sample1 和$sample2 之间的关系看成是一个人在照镜子，这个人做的任何动作，镜子里的他都会重复得一模一样，这就是“赋值”的内涵。而在对象$sample1 和$sample3 之间存在的却不是这种关系，它们更像是一对双胞胎：有着惊人相似的外貌，却有着绝对独立的属性，这才是“克隆”的内涵。

12.3.6　方法链

方法链是 PHP 5 中引入的一种让我们可以在对象被返回的同时使用该对象的方法和属性，就像下面这样：

```
$someObject -> getObjectOne () -> getObjectTwo() -> callMethodOfObjectTwo();
```

在上面这个示例中，有一个名为$someObject 的对象，用 getObjectOne 的方法来获取名为 objectOne 的对象。而 objectOne 对象用 getObjectTwo 方法来获取名为 objectTwo 的对象。而 objectTwo 的对象有一个名为 callMethodOfObjectTwo 方法用来执行最后的步骤。

在这种情况下，我们可以使用上述链式引用的方法来执行 objectTwo 对象的 callMethodOfObjectTwo 方法。

现在我们来看一个实际的例子。

【例 12.20】 定义名为 emailBuilder 的类（\\class.emailbuilder.php）。

```
<?php
    //定义名为 methodChain 的类
    class emailBuilder {
        //定义一些私有变量
        private $from;
        private $to;
        private $subject;
        private $body;
        private $cc;
        private $date;

        //初始化 emailBuilder 的时间变量
        function   construct(){
            $this -> date = date('Y 年 m 月 d 日');
        }

        //设置发件人
        function from ($from){
            $this -> from = $from;
            return $this;
        }

        //设置收件人
        function to ($to) {
            $this -> to = $to;
            return $this;
        }

        //设置邮件主题
        function subject ($subject) {
            $this -> subject = $subject;
            return $this;
        }

        //设置邮件内容
        function body ($body) {
            $this -> body = $body;
            return $this;
        }

        //设置邮件抄送对象
        function cc ($cc) {
            $this -> cc = $cc;
            return $this;
        }

        //输出邮件
        function buildEmail() {
            echo '<b>>>发件人: </b>'.$this -> from.'<br>'.
                '<b>>>>收件人: </b>'.$this -> to.'<br>'.
                '<b>>>>抄送: </b>'.$this -> cc.'<br>'.
                '<b>>>>日期: </b>'.$this -> date.'<br>'.
                '<b>>>>主题: </b>'.$this -> subject.'<br>'.
                '<b>>>>内容: </b>'.'<br><b>>>></b>'.$this -> body;
        }
```

```
    }
?>
```

在这个名为 emailBuilder 的类中，我们定义了一些私有变量做为邮件的组件。然后再定义了一些公有方法，用来修改这些组件内容。最后定义了一个名为 buildEmail 的公有方法，用来输出整封邮件的内容。

在这里，需要提醒注意的是，所有用来修改邮件组件的方法返回的值都是$this，而这才是使用方法链的关键。

【例 12.21】 使用方法链定义并输出邮件内容（\\Ex12-21.php）。

```
<?php
    include_once('class/class.emailbuilder.php');

    //定义了一个名为$eb 的 emailBuilder 对象
    $eb = new emailBuilder();

    //使用方法链定义并输出邮件内容
    $eb -> from('sender@example.com')
        -> to('receiver@example.com')
         -> cc('ccto@example.com')
         -> subject('这是一封用 emailBuild 类创建的邮件')
         -> body('要使用 emailBuilder，先创建一个基于 emailBuilder 类的对象，
         然后使用方法链定义并输出邮件内容。')
         -> buildEmail();
?>
```

在上面这段例程中，我们定义了一个基于 emailBuilder 类的对象$eb，然后使用方法链设置并输出了邮件的内容，如图 12-6 所示。

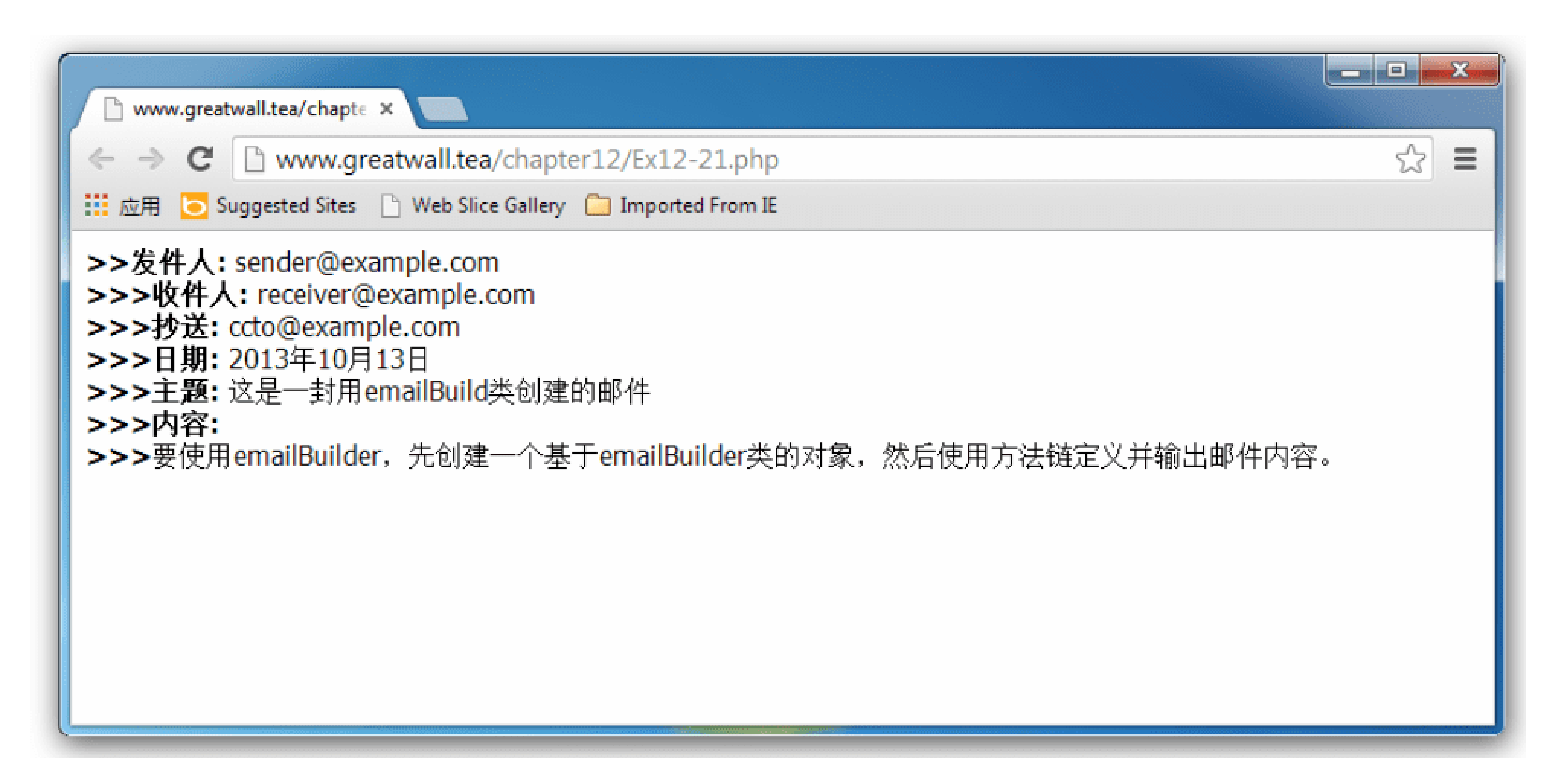


图 12-6　使用方法链设置并输出邮件内容

12.4　设 计 模 式

基于对象的编程，其目的是为了简化编程过程、减少开发时间和简洁脚本代码。如果

使用得当，基于对象的编程可以极大地提升脚本的性能。早在 1972 年，Eric Gamma 和他的三个朋友就针对基于对象的编程提出了一系列的基于对象编程的标准化模式，这些标准化模式被称为“设计模式”，而这四个人则被人形象地称为“四人帮（Gang of Four）”。

看到这个极度抽象的名词，很多同学都会不由自主地产生畏难的心理。因为我当初接触这个“设计模式”的时候，也有相同的想法。不过，大家再想一想，如果设计模式是为了减缓编码效率和脚本执行效率，肯定没有人用。所以，它的首要目的肯定是提升编码效率和脚本执行效率。为了实现这个目的，“四人帮”在他们常年地编码实践中总结并抽象出了一些最佳实践，形成了一些标准化的编码模式。

其实，大家在编码的过程中，肯定也用到了一些与这些最佳实践类似的提升编码效率和减少编码量的方法。只是不知道这些方法可以被抽象到“模式”这么一个高度。

在本节里，我们就来简单地了解一下在 PHP 编码过程中常常会用到的一些设计模式。

12.4.1　策略模式（Strategy）

策略模式通常用在某个算法需要有多个不同的变体以应对不同情况的时候。打个比方，如果你需要定义一个类用来创建图片。但是图片有很多的格式，在某些情况下，想使用该方法创建 JPEG 格式的图片；而在另一些情况下，想使用该方法创建 GIF 格式的图片。则可以把这个类定义成下面这个样子：

```
<?php
    class createImage {
        private $images;

        function createJPEG() {
            $this -> images = 'image.jpg';  //此处代码只用于示例，不能创建 JPEG
                                              格式的图片
            return $this -> images;
        }

        function createGIF() {
            $this -> images = 'image.gif';  //此处代码只用于示例，不能创建 GIF
                                              格式的图片
            return $this -> images;
        }
    }
?>
```

在这个类中，可以为每一种需要的图片格式定义一个 creator 方法。但是一旦需要的图片格式比较多，这个类就会被撑到很大。到时候，不光是脚本难以维护，脚本的性能也会大打折扣。为了解决这个问题，我们可以先定义一个抽象类（abstract），然后再定义基于这个抽象类的针对不同格式的子类来具体地创建指定格式的图片。这样一来，类的数量虽然增加了，但是不仅维护起来要容易的多，而且脚本性能也会得到提升。如果需要新的图片格式，再创建一个基于抽象类的新类就好了。

按照这样的想法，上面这段脚本就可以修改成下面这个样子：

```
<?php
    abstract class createImage {
        protected $image;
```

```
        function createImage(){
        }
    }

    class createJPEG extends createImage {

        function createImage(){
            $this -> image = 'image.jpeg';  //此处代码只用于示例，不能创建 JPEG
                                              格式的图片
            return $this -> image;
        }
    }

    class createGIF extends createImage {

        function createImage(){
            $this -> image = 'image.gif';  //此处代码只用于示例，不能创建 GIF
                                             格式的图片
            return $this -> image;
        }
    }
?>
```

12.4.2　工厂模式（Factory）

在策略模式一节里，我们定义了一个抽象类，然后从这个抽象类中派生出了两个具体类。如果我们直接使用这两个具体类来创建对象，然后使用该对象的 creator 方法创建图片的话，那这个抽象类的存在就没有意义了。

通常情况下，我们在使用策略模式定义了一些策略后，会使用工厂模式来执行这些策略。换句话说，策略模式就像是工厂的高层制订的生产策略，而工厂模式是车间主任根据在生产过程中遇到的不同情况在这些策略中选择具体的策略，以保证生产的进行。所以工厂模式通常是和策略模式一起使用的。

按照这个思路，我们可以在上面这段脚本末尾加上一个名为 imageFactory 的类，来具体执行创建图片的过程。这个 imageFactory 类的脚本如下：

```
class imageFactory {
    private static $format;

    static function creator ($format) {
        if (!isset(self::$format)) {
            self::$format = $format;
        }

        switch (self::$format) {
            case 'JPEG':
                return new createJPEG();
                break;
            case 'GIF':
                return new createGIF();
            default:
                return '无法创建图片。没有找到指定的图片格式对应的类。';
        }
    }
}
```

在这个 imageFactory 类中，我们定义了一个静态私有属性$format 用来存放所需图片的格式。接着，定义了一个静态的名为 creator 的方法，该方法会根据私有属性的值来判断需要创建的对象。如果所需图片的格式为 JPEG，在使用 imageFactory 类的 creator 方法时，指定其 format 参数为“JPEG”，那么 creator 方法返回的就是一个 createJPEG 实例，若指定其 format 参数为“GIF”，那么 creator 方法返回的就是一个 createGIF 实例。由于这两个实例拥有相同的方法和属性，我们可以将 creator 方法返回的对象存储到一个变量中，然后通过这个变量来使用 createJPEG 和 createGIF 类的方法，就像下面这样：

```
$format = 'JPEG';

$creator = imageFactory::creator($format);

if(!is string($creator)){
    echo $creator -> createImage();
} else {
    echo $creator;
}
```

12.4.3　单体模式（Singleton）

单体模式通常用在某个类只需要被实例化一次不允许被再次实例化的情况下。比如，我们在定义上面这个 imageFactory 类的时候，就使用了单体模式。在讲解什么是单体模式之前，先把 12.4.2 小节末尾的那段脚本修改成如下：

```
$format = 'JPG';

$creator = imageFactory::creator($format);

$format = 'JPEG';

$creator = imageFactory::creator($format);

if(!is string($creator)){
    echo $creator -> createImage();
} else {
    echo $creator;
}
```

在这段脚本中，我们定义了一个名为$format 的变量，并为其赋值“JPG”，然后将这个变量代入 imageFactory::creator($format)中，想要创建一个用来创建 JPG 格式图片的 creator 对象。我们知道，之前并没有定义用来创建 JPG 格式图片的 creator 类，所以这里应该输出的是如下所示报错信息：

```
无法创建图片。没有找到指定的图片格式对应的类。
```

接着，将变量$format 的值修改为“JPEG”，然后再次使用 imageFactory::creator($format)实例化 imageFactory 类，以创建一个用来创建 JPEG 格式图片的 creator 对象。我们知道，之前已经定义了用来创建 JPEG 格式图片的 creator 类，所以这里应该可以返回一个基于 createJPEG 类的对象。

可出乎我们意料的是，这段脚本执行的结果依然是上面那条报错信息。

为什么呢？

因为我们在 imageFactory 类中定义其属性和方法时，使用了 static 修饰词，这就使得 imageFactory 类中的属性和方法在 imageFactory 类第一次被实例化后就固定了下来，不会再发生变化了。换句话说，通过使用 static 修饰词实现了单体模式，或者说，使用单体模式定义一个类，就是通过为该类创建静态属性和方法，来实现该类只会被实例化一次的目的。

12.4.4　观察员模式（Observer）

在 PHP 应用程序中，通过对数据的操控来实现许多功能。在很多情况下，某个数据的改变会影响到该程序的很多方面。以购物网站上的商品列表页面为例：当一个位于中国的消费者打开该页面时，商品的价格是以人民币计价的；而当一个位于美国的消费者打开该页面时，商品的价格则是以美元计价的。在这两次访问中，只有一个参数不一样，那就是用户发起访问的地点。也就是说，用户发起访问的地点这个参数影响到了商品列表页面上显示的商品价格。一旦用户发起访问的地点发生了变化，商品列表上展示的商品的价格也会发生变化。

我们可以把商品列表上展示的每件商品当成是一个观察员对象。它们作为观察员的职责是盯住用户发起访问的地点这个参数，一旦这个参数发生了变化，立即更新其价格属性并输出。

为了实现这个功能，我们可以使用观察员模式。先创建一个名为 accessLocObserver 的接口类，在其中定义一个 setPrice 方法，就像下面这样。

【例 12.22】 创建一个观察员接口类（\\interface.accesslocobserver.php）。

```
<?php
    //创建一个名为 accessLocObserver 的接口类，用来观察 accessLocation 类的变化
    interface accessLocObserver {

        //一旦 accessLocation 类中发生了变化，就设置物品的价格属性
        function setPrice($obj);
    }
?>
```

在这个接口类中，定义了一个 setPrice 方法，该方法有一个参数，就是由被观察的类派生出来的对象。

接着，创建一个名为 accessLocation 的类，该类拥有一个名为$location 的私有属性。一旦该属性被重新设置，就会触发 notifyObservers 方法。另外，该类还有一个静态属性 $instance 和静态方法 getInstance，可供我们在实现 accessLocObserver 类的具体类中通过“::”实例化 accessLocation 类，并使用其拥有的方法。除此之外，accessLocation 类还有一个名为 observers 的私有属性，用来存储注册到该类的所有对象；一个名为 getLocation 的方法，用来获取当前用户发起访问的地点；一个名为 registerObserver 的方法，用来将指定的对象注册到由 accessLocation 类派生出的对象中；一个名为 notifyObservers 的方法，用来通知已经注册到由 accessLocation 类派生出的对象中的所有对象。

这个名为 accessLocation 的类如下。

【例 12.23】 创建被观察员观察的类 accessLocation（\\ class.accesslocation.php）。

```
<?php
    //创建一个名为 accessLocation 类
```

```
    class accessLocation {

        //创建一个私有属性 instance，用来在类中实例化自身
        static private $instance = NULL;

        //创建一个私有属性，用来存储用户发起访问的地点
        private $location = "中国大陆";

        //创建一个私有数组，用来存储注册到 accessLocation 类的观察员
        private $observers = array();

        //创建一个静态方法，用来在类中实例化自身
        static function getInstance() {
            if(self::$instance == NULL) {
                self::$instance = new accessLocation();
            }

            return self::$instance;
        }

        //创建用来获取当前用户发起访问地点的方法
        function getAccessLocation() {
            return $this -> location;
        }

        //创建用来设置当前用户发起访问地点的方法
        function setAccessLocation($newLocation) {
            $this -> location = $newLocation;
            //在设置当前用户发起访问地点的同时，通知所以注册到该类的观察员
            $this -> notifyObservers();
        }

        //创建用来将观察员注册到 accessLocation 类的方法
        function registerObservers($observer) {
            $this -> observers[] = $observer;
        }

        //创建用来通知观察员的方法
        function notifyObservers(){
            foreach ($this -> observers as $observer) {
                $observer -> setPrice($this);
            }
        }
    }
?>
```

在上面的脚本中，值得我们注意的是，在 setAccessLocation 方法中除了设置私有属性 $location 的值之外，还触发了 notifyObservers 方法。

接下来，还要定义一个用来实现 accessLocObserver 接口类的方法，并将其命名为 productItem。

【例 12.24】 创建 productItem 类（\\class.productitem.php）。

```
<?php
    include once('interface/interface.accesslocobserver.php');
    include once('class/class.accesslocation.php');

    //创建用来实现 accessLocObserver 接口类的商品类
    class productItem implements accessLocObserver {
```

```
        private $price;
        private $forexRates = array ('美国' => 0.1634,
                                     '欧洲' => 0.1206,
                                     '英国' => 0.1024,
                                     '中国香港' => 1.2669,
                                     '中国台湾' => 4.7992,
                                     '澳大利亚' => 0.1726,
                                     '日本' => 16.1074,
                                     '韩国' => 175.0426);

        //初始化 productItem 类并将其注册到 accessLocation 类中
        function __construct($price){
            $this -> price = $price;
            accessLocation::getInstance()->registerObservers($this);
        }

        //定义 accessLocObserver 接口类的 setPrice 方法接口的实现
        function setPrice($obj){
            if($obj instanceof accessLocation) {
                $this -> price = $this -> price *
                $this -> forexRates[accessLocation::getInstance() ->
                getAccessLocation()];
            }
        }

        //定义获取产品当地价格的方法
        function getPrice(){
            switch (accessLocation::getInstance() -> getAccessLocation()) {
                case '美国':
                    $sign = "$ ";
                    break;
                case '欧洲':
                    $sign = "€ ";
                    break;
                case '英国':
                    $sign = "£";
                    break;
                case '中国香港':
                    $sign = "$ ";
                    break;
                case '中国台湾':
                    $sign = "NT$ ";
                    break;
                case '澳大利亚':
                    $sign = "$ ";
                    break;
                case '日本':
                    $sign = "¥";
                    break;
                case '韩国':
                    $sign = "₩ ";
                    break;
                default:
                    $sign = "¥";
                    break;
            }
```

```
            return accessLocation::getInstance() -> getAccessLocation().
                '售价：'.$sign.number_format(floatval($this->price),2);
        }
    }
?>
```

在这个名为 productItem 的类中，我们为其定义了两个私有属性，分别为$price 和$forexRates。在初始化这个类时，设置了私有属性$price 的值，并将该类注册到了由 accessLocation 类派生出来的对象中。然后，在 productItem 类中分别定义了一个 setPrice 和一个 getPrice 方法，前者是我们在 accessLocObserver 接口类中规定的必须实现的方法，用来根据 accessLocation 对象的私有属性$location 的值的不同而设置不同的价格；而 getPrice 是我们自行定义一个用来获取当前价格的方法。

最后，我们要用 accessLocation 和 productItem 两个类来实现本节开始的时候描述的功能，即一旦用户发起访问的地点发生了变化，商品列表上展示的商品的价格也会发生变化。

【例 12.25】 商品售价随用户访问地点的不同而发生变化（\\Ex12-25.php）。

```
<?php
    include_once('class/class.accesslocation.php');
    include_once('class/class.productitem.php');

    $pi = new productItem(1588);
?>
<!DOCTYPE html>
<html>
    <head>
        <title>随地点变化的商品售价</title>
    </head>
    <body>
        <form action="?location" method="post">
            访问地点
            <select name="location">
                <option value="美国">美国</option>
                <option value="欧洲">欧洲</option>
                <option value="英国">英国</option>
                <option value="澳大利亚">澳大利亚</option>
                <option value="日本">日本</option>
                <option value="韩国">韩国</option>
                <option value="中国香港">中国香港</option>
                <option value="中国台湾">中国台湾</option>
            </select>
            <input type="submit" value="显示售价">
        </form>
        <hr />
<?php
    //输出商品的初始价格
    echo $pi -> getPrice()."<br>";

    //判断用户是否提交了表单
    if(isset($_REQUEST['location'])){

        //若用户提交了表单，则根据用户的选择输出对应的结果
        $location = $_REQUEST['location'];
        switch ($location) {
            case '美国':
                accessLocation::getInstance() -> setAccessLocation('美国');
```

```
                    echo $pi -> getPrice()."<br>";
                    break;
                case '欧洲':
                    accessLocation::getInstance() -> setAccessLocation('欧洲');
                    echo $pi -> getPrice()."<br>";
                    break;
                case '英国':
                    accessLocation::getInstance() -> setAccessLocation('英国');
                    echo $pi -> getPrice()."<br>";
                    break;
                case '澳大利亚':
                    accessLocation::getInstance()-> setAccessLocation('澳大利亚');
                    echo $pi -> getPrice()."<br>";
                    break;
                case '日本':
                    accessLocation::getInstance() -> setAccessLocation('日本');
                    echo $pi -> getPrice()."<br>";
                    break;
                case '韩国':
                    accessLocation::getInstance() -> setAccessLocation('韩国');
                    echo $pi -> getPrice()."<br>";
                    break;
                case '中国香港':
                    accessLocation::getInstance()-> setAccessLocation('中国香港');
                    echo $pi -> getPrice()."<br>";
                    break;
                case '中国台湾':
                    accessLocation::getInstance()-> setAccessLocation('中国台湾');
                    echo $pi -> getPrice()."<br>";
                    break;
            }
        }
?>
    </body>
</html>
```

在这个页面上，从 productItem 类中派生出了一个名为$pi 的对象，并在页面上输出了$pi 对象的初始价格。然后根据用户的选择输出对应地区的商品价格。

在脚本中，值得我们注意的是，通过 accessLocation::getInstance 方法来使用 accessLocation 实例化后的静态方法。由于 getInstance 返回的是 accessLocation 的一个实例，因此，我们可以直接在其后面链式使用 setAccessLocation 方法，从而根据用户的选择设置当前 accessLocation 对象的私有属性$location 的值。

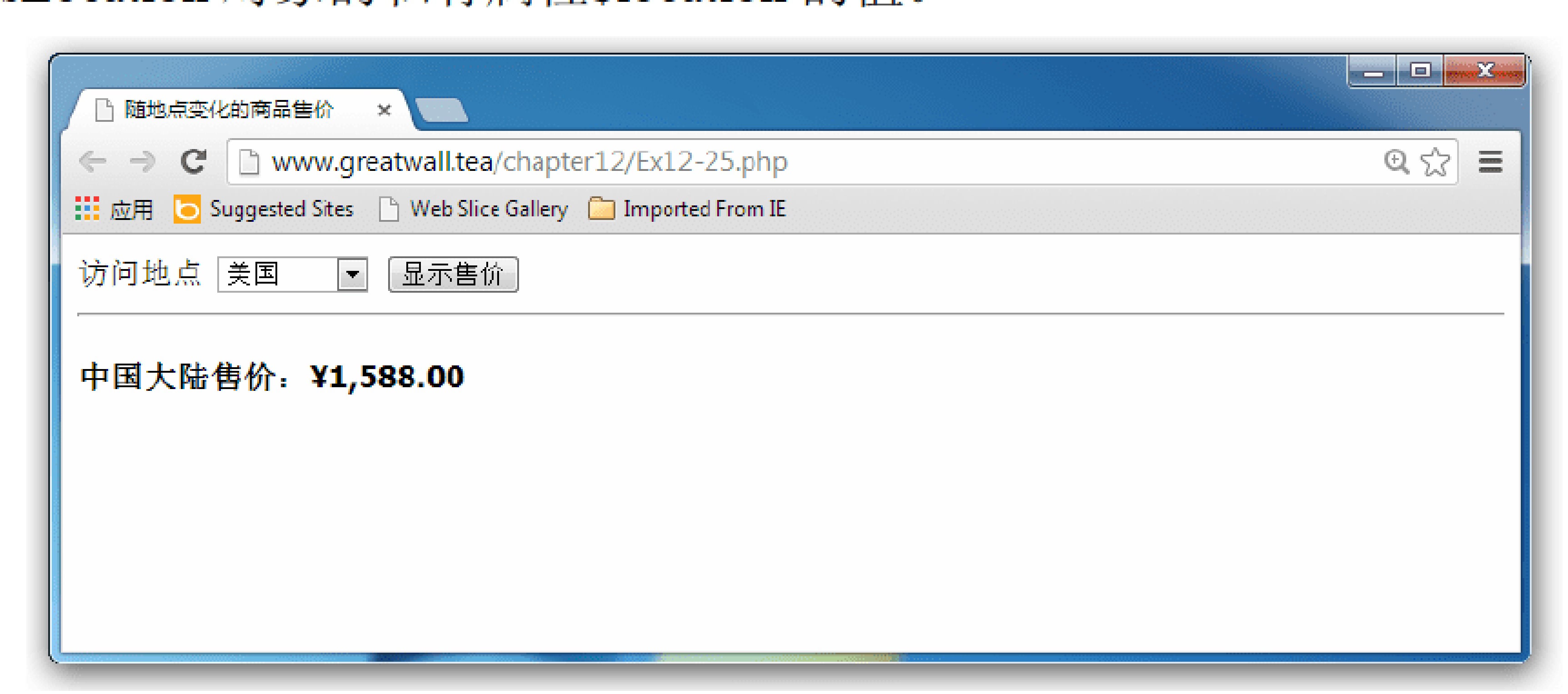


图 12-7　页面输出的原始价格

现在我们直接单击图 12-7 页面上的“显示售价”按钮，页面上显示的内容如图 12-8 所示。

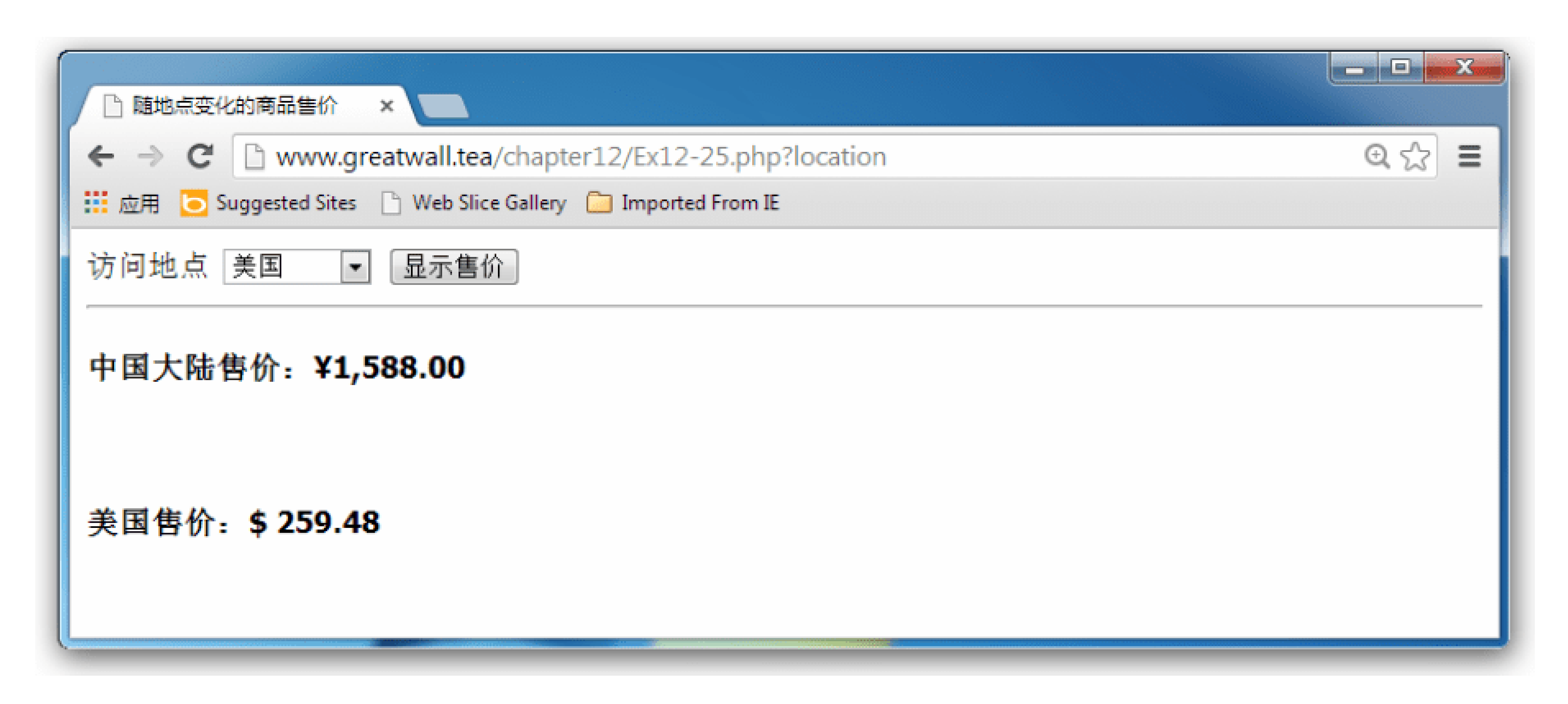


图 12-8　页面输出的美国当地价格

12.5　习　　题

（1）请使用策略模式编写一个抽象类 createFile。然后创建两个派生自 createFile 类的子类，分别为 createTXT 和 createCSV，用于创建文本文件和 CSV 文件。

（2）请使用工厂模式编写一个抽象类 fileFactory，用于创建文本文件和 CSV 文件。

第 13 章　PHP 与 MVC

在上一章里，我们学习了如何使用基于对象编程的方法来缩写 PHP 脚本，也了解了如何使用一些常用的设计模式来简化我们日常的编码工作。如果我们能够从一个更为宏观的视角来看 PHP 应用程序的开发过程，应该从应用程序的架构谈起。在本章，会了解一种非常流行的应用程序架构，它被称为 MVC，是 Model-View-Controller 的简称。

在 PHP 快速应用开发中，MVC 扮演了十分重要的角色。现如今，市场上有着形形色色的基于 MVC 架构的开发框架。在它们当中，使用比较广泛的开发框架有 CakePHP、CodeIgniter 和 Zend Framework 等等。它们虽然不全是基于对象编程的框架，但是却都是基于 MVC 架构的应用框架。使用这些框架可以极大地加快应用程序的开发。

在这样的背景下，白手起家写应用程序的日子似乎已经到头了，因为更多的应用程序是基于开发框架的。如果想要更好地了解 PHP 应用程序的开发过程，就必须了解一下什么是 MVC。

13.1　MVC 大起底

在本小节，我们来具体了解一下 MVC 架构是什么。

13.1.1　什么是 MVC

前面说到，MVC 是 Model-View-Controller 的简称。顾名而思义，所谓的 MVC 架构有着三个重要的组成部分，它们分别是模型（Model）、视图（View）和控制器（Controller）。那么这三位又都是干什么的呢？

简单地说，模型（Model）是数据库和第三方服务互动的对象，它定义了应用程序的业务逻辑。所谓业务逻辑，指的是任何与通过存储数据和使用第三方服务来达到业务需求相关的东西。如果应用程序需要访问数据库，那么为应用程序添加功能的代码就应该被写入模型对象中。

视图（View）则是所有与信息呈现有关的元素的集合，通常情况下，视图文件是一些 HTML、CSS 和 JavaScript 文件。换句话说，所有用户看到的和与之互动的页面都是视图页面。一个视图页面只有在控制器与模型互动产生了用户需要的结果后才会被调用，并通过视图将结果返回给用户。

控制器（Controller）则是用来联系用户、模型和视图的中间部件，它隔离并沟通了业务逻辑和结果呈现。通常情况下，控制器是用户访问的起始点，因为所有的用户请求都由

控制器处理。当控制器接收到用户请求时，会根据用户请求调用模型方法，通过模型获取用户所需的数据，然后调用视图文件，将用户所需的数据通过视图文件返回给用户。图 13-1 展示了模型、视图和控制器的互动过程。

在图 13-1 中，用户向控制器发出请求，控制器根据用户请求调用相应的模型，并将模型返回的用户所需结果通过视图返回给用户。

需要注意的是，该图只是一个原理示意图。在实际的应用程序开发中，并不是所有的用户请求都需要调用模型或视图的。有的用户请求涉及到的模型可能非常复杂，一个模型解决不了问题，这时就需要调用多个模型，而其中肯定会有些模型做为中间件而不需要调用视图。而有的时候，则完全不需要框架，直接由控制器将用户需求转向视图，继而将内容呈现给用户。

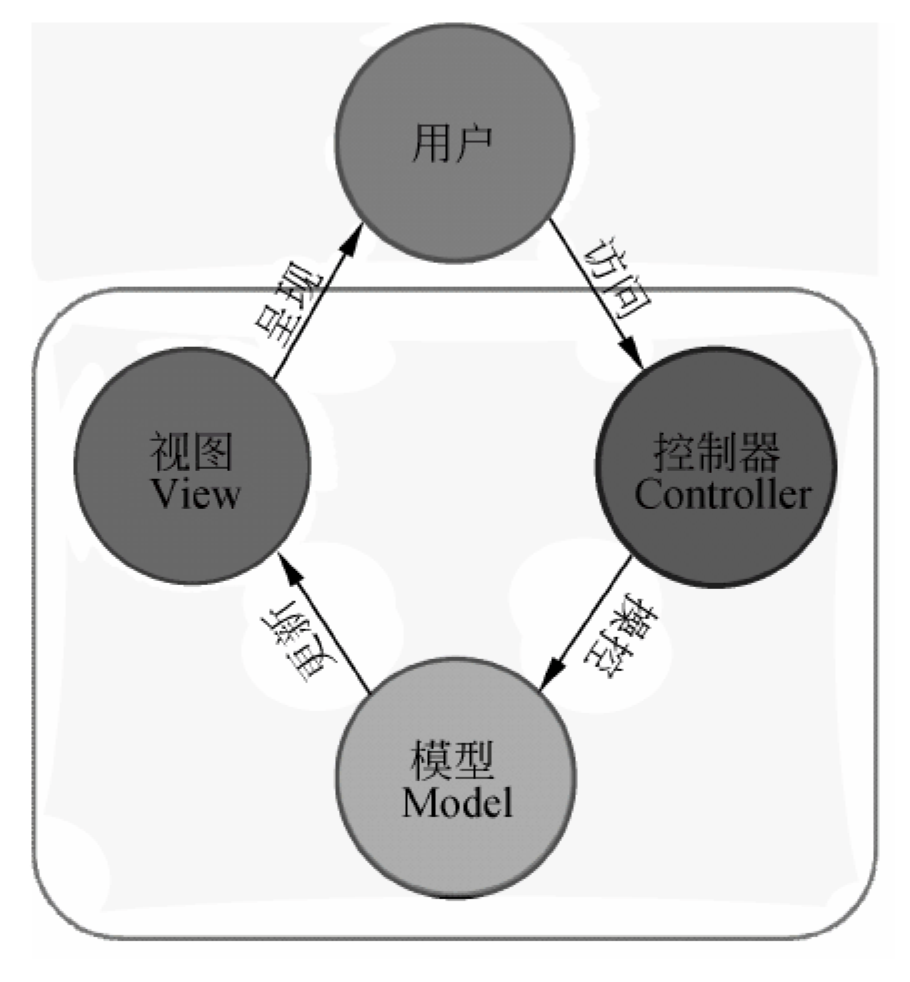


图 13-1　模型、视图和控制器

13.1.2　为什么要使用 MVC

其实通过图 13-1，大家应该可以猜到为什么我们要使用 MVC 架构。如果你还没有猜透，那再看一遍 MVC 架构的工作流程：当控制器接收到用户发起的访问请求后，会根据用户的请求通过模型获取用户请求的数据；在取得用户请求的数据后，将数据通过视图展示给用户。控制器、模型和视图就像一个团队中紧密配合的成员，各司其职，实现共同的目标。在这个过程，业务逻辑、数据读写和内容呈现被分隔成了三个独立而又相互联系的部件。

使用 MVC 模型，我们可以并行开发应用的业务逻辑、数据读写和内容呈现部件，加快开发进程。同时，由于这三个部分相互独立，我们对其中某个部件的修改完全不会对其他两个部件造成影响。由于这两个显而易见的好处，使用 MVC 模型进行 PHP 应用程序的开发成了我们的不二选择。

13.1.3　常用的 MVC 框架

为了进一步加快应用程序的开发，市场上已经有很多成熟的开发框架供我们选择。在这些开放框架中，有不少还是开源的框架。使用这些开发框架，就可以站在巨人的肩膀上开始我们的开发工作，获得事半功倍的效果。

下面就来了解一下这些开源的 MVC 开发框架。

（1）CodeIgniter

CodeIgniter 框架是世界上第一个，也是最简洁的一个基于 MVC 模型的 PHP 应用程序开发框架。它是由 EllisLab 开发和维护。虽然 EllisLab 将 CodeIgniter 定义为一个开源的 MVC 开发框架，不过其对 CodeIgnite 的使用还是有着严格的控制。

CodeIgniter 有着完善的开发文档和高质量的代码，甚至于有些开发框架，比如 KohanaPHP，深受 CodeIgniter 框架的影响。虽然 CodeIgniter 在流行程度上可能比不上下面介绍的一些开发框架，这完全是由于 ElliseLab 对 CodeIgniter 的管控策略造成的，不过

EllisLab 正在对这一问题做出改变。

（2）Zend Framework

Zend Framework 是一套可扩展、轻耦合的脚本库。这套基于 MVC 架构的脚本库可以做为 PHP 应用程序的基础架构以加快应用程序的开发。该框架由 Zend Technologies 开发并维护，适用于大型应用程序的开发。

（3）CakePHP

CakePHP 是目前最受欢迎的基于 MVC 架构的 PHP 应用程序开发框架。与前两个框架不同的是，CakePHP 的开发与维护不是由某一个组织完成的，它有着庞大的开发者社群和广泛的群众基础。

在 CakePHP 开发框架中，规约性是首要的。只有当控制器、模型和视图的命名完全符合规范时，它们才会有机的结合起来。

13.2　KISSMVC：一个简单的 MVC 框架

在上一节里，我们知道了什么是 MVC 架构和为什么要使用 MVC 架构进行应用程序的开发。另外，我们还了解了当前市场上可供选择的一些基于 MVC 架构的 PHP 应用程序开发框架。当使用这些开发框架进行应用程序的开发时，除了需要掌握 PHP 脚本的相关知识和开发技能之外，还得了解这些框架的内在机制。有些框架很简单、上手也很容易；而有些框架的学习曲线就很长，需要我们投入大量的时间和精力。

为了能更快地了解基于 MVC 的 PHP 应用程序开发，在本节里，我们将解剖一个简单且开源的基于 MVC 的开发框架——KISSMVC。据 KISSMVC 发布网站上的介绍，KISSMVC 框架是这个世界上体积最小、且最简单的基于 MVC 的 PHP 应用程序开发框架。其代码简单易读，几分钟之内就能够掌握其核心功能。

13.2.1　KISSMVC 框架概述

KISSMVC 框架的核心文件只有一个，而这个文件也仅有 16 KB 大小。该文件为我们提供了三个抽象类，分别如下。

- KISS_Model 类：该类是一个简单的基于对象关系映射（ORM）的数据库操控类。在 KISSMVC 框架中，主要用于操控存储在关系型数据库中的数据。
- KISS_View 类：该类是一个简单的模板系统。在 KISSMVC 框架中，负责数据的呈现。
- KISS_Controller 类：该类是 KISSMVC 框架的中间件。可以解析用户请求、并根据用户请求调用相应的基于 KISS_Model 和 KiSS_View 类的子类和对象。

在本小节中，为了便于开发，我们将直接从 KISSMVC 框架的发布网站（http://kissmvc.com）下载名为“KISSMVC Blank Project”的压缩包文件。然后将下载下来的压缩包解压放到 Apache 服务器的根目录下的名为 chapter13 的子目录中。

解压后的目录结构如下：

```
|------chapter13
|------app
|------controllers
|------main
|------index.php
|------helps
|------inc
|------models
|------views
|------layout.php
|------css
|------image
|------js
|------.htaccess
|------index.php
|------kissmvc.php
|------kissmvc_core.php
```

在这个解压后的目录中，核心文件为 kissmvc_core.php，上文提到的三个抽象类就在这个文件中，后续我们会对该文件中的内容进行详细地解读。

在该目录下，我们还有一个名为 kissmvc.php 的文件。在这个文件中，定义了三个派生自 kiss_mvc.php 中的三个抽象类的普通类，后续在创建控制器、模型和视图对象都是基于这三个普通类的。这样做的好处是可以防止 kissmvc_core.php 中的抽象类被篡改，同时也方便了我们在普通类中定义新的通用方法和属性。

我们还有一个名为 index.php 的文件，这个文件是与用户交互的唯一入口。基于 KISSMVC 框架的 PHP 应用程序就是通过该文件与用户互动的。

除此之外，在 kissmvc 目录下，还有一个重要的名为 app 的子目录。该目录为应用目录，PHP 应用程序的所有内容都需要放在 app 目录下相应的子目录中。其中子目录的内容如下。

- Controllers 目录：用来存放所有的控制器文件；
- Models 目录：用来存放所有的模型文件；
- Views 目录：用来存放所有的视图文件。

关于 kissmvc 目录下的其余文件和文件夹的作用，我们会在用到它们的时候再进行简单地介绍。

当我们把下载下来的压缩包解压好之后，将框架目录下的 index.php 文件中的 WEB_FOLDER 常量由之前的“/kissmvc/”改成“/chapter/”。然后打开浏览器，访问 chapter13 目录，会看到如图 13-2 所示的内容：

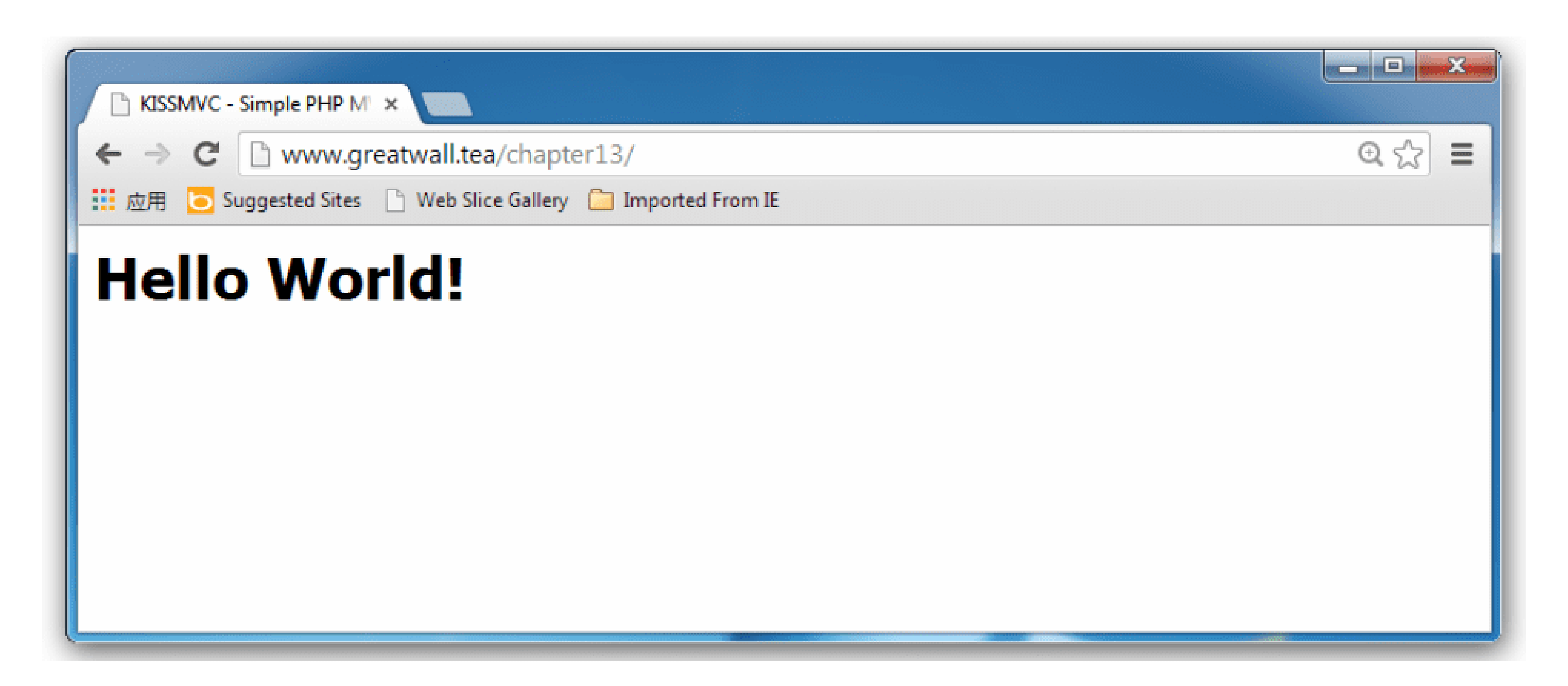


图 13-2　基于 KISSMVC 框架开发 PHP 应用程序项目的初始页面

另外需要说明的是，KISSMVC 框架支持通过分布式配置文件[i]（.htaccess）改造 URL。在第 9 章里，我们学习了如何通过 URL 在页面间传递参数。以 http://www.example.com/users.php?id=a_florrick 为例，id=a_florrick 就是我们通过 URL 传递的参数。但是这样的 URL 对于搜索引擎来说是不太友好的。另外，想让人记住这样的 URL 也比较困难。

如果我们用 Apache 服务器做为 Web 服务器并在 Apache 服务器上开启 mod_rewrite 模块，那么就可以在分布式配置文件（.htaccess）里添加相应的 URL 改写规则，从而实现通过类似于 http://www.example.com/main/users/a_florrick/这样的 URL 来访问这个页面的目的。大家可以比较一下这两个 URL，哪一个看上去更容易理解呢？

13.2.2　框架入口（index.php）

在上一节里，我们提到框架根目录下的 index.php 文件是用户和框架交互的唯一入口。这就意味着，用户所有的请求都会通过 index.php 文件传递给相应的控制器。那么，我们就先来看看 index.php 文件里有些什么内容。

在 index.php 文件开始，开启了应用程序调试功能并通过.htaccess 文件定义了 URL 改写规则。然后，通过下面两行脚本定义了 KISSMVC 框架的两个核心常量：一个是应用程序所在路径，而另一个则是框架所在路径。它们分别对应框架下的“app”目录和框架所在目录。

```
//===============================================
//Madatory KISSMVC Settings (please configure)
//===============================================
define('APP_PATH','app/'); //with trailing slash pls
define('WEB_FOLDER','/chapter13/'); //with trailing slash pls
```

接着通过自动化全局变量[ii] $GLOBALS 定义了网站的标题。

```
//===============================================
//Other Settings
//===============================================
$GLOBALS['sitename']='KISSMVC - Simple PHP MVC Framework';
```

然后使用 require()函数导入了 kissmvc.php 文件。在该文件中，定义了三个派生自框架核心文件中的三个抽象类的普通类。这三个普通类分别是 Controller、Model 和 View。

```
//===============================================
//Includes
//===============================================
require('kissmvc.php');
```

最后定义了一个基于 Controller 类的对象：

```
//===============================================
//Start the controller
//===============================================
$controller = new Controller(APP_PATH.'controllers/',WEB_FOLDER,'main',
'index');
```

通过这四个部分，我们构建了应用程序的访问入口。在这四个部分中，最为关键的是

---

[i] 关于分布式配置文件的详细介绍，可以参考 Net Tutsplus 网站上的一篇名为“An In Depth Guide to mod_rewrite for Apache”的文章（http://net.tutsplus.com/tutorials/other/a-deeper-look-at-mod_rewrite-for-apache/）。

[ii] 所谓自动化全局变量，指的是在脚本所有作用域中都起作用的变量。

定义了一个基于 Controller 类的名为$controller 对象。正是通过该对象，我们才能看到如图 13-2 所示的页面。

13.2.3　控制器（KISS_Controller）

我们刚刚讲到，在 index.php 文件中，定义了一个基于 Controller 类的名为$controller 的对象。通过该对象，连通了 KISSMVC 框架的控制器、模型和视图，完成了在页面上呈现用户所请求的数据的任务。

在本小节里，我们就来详细了解一下 KISSMVC 框架的控制器类——KISS_Controller。

在 KISS_Controller 类中，我们主要实现了如下三个功能：

- ❑ 通过 explode_http_request 和 parse_http_request 方法解析用户请求，分离出所需的控制器和行为的名称；
- ❑ 通过 route_request 方法调用所需的控制器和行为；
- ❑ 通过 request_not_found 方法在无法找到所需控制器和行为时返回“404 无法访问”页面。

在下面的脚本中，定义了六个受保护的属性，分别用来存放控制器文件路径（$controller_path）、框架路径（$web_folder）、用户请求信息（$request_uri_parts）、控制器名称（$controller）、行为名称（$action）和用户通过 URL 传递的参数（$params）。随后，我们使用__construct 魔术方法初始化了这些受保护的属性并导入相应的控制器文件。

```
//===============================================================
//控制器类
//解析用户请求，定位相应控制器。
//===============================================================
abstract class KISS_Controller {
  protected $controller_path='../app/controllers/'; //with trailing slash
  protected $web_folder='/'; //with trailing slash
  protected $request_uri_parts=array();
  protected $controller;
  protected $action;
  protected $params=array();

  function __construct($controller_path,$web_folder,$default_controller,
  $default_action)  {
    $this->controller_path=$controller_path;
    $this->web_folder=$web_folder;
    $this->controller=$default_controller;
    $this->action=$default_action;
    $this->explode_http_request()->parse_http_request()->route_request();
  }

  //该方法用于解析用户请求，并将包含用户请求信息的字符串存入属性 request_uri_parts 中
  function explode_http_request() {

  }

  //该方法用于从属性 request_uri_parts 中分离出响应用户请求的控制器、行为和用户通过
URL 传递的参数
  function parse_http_request()  {

  }
```

```
  //该方法根据分离出来的控制器和行为名称查找并导入相应的控制器和行为文件
  function route_request() {

  }

  //该方法定义了在无法找到指定的控制器时输出的 404 页面
  function request_not_found($msg='') {

  }
}
```

下面，我们来详细看看 KISS_Controller 类中定义的四个方法。

（1）解析用户请求（explode_http_request 和 parse_http_request）

```
function explode_http_request() {
  $requri = $_SERVER['REQUEST_URI'];
  if (strpos($requri,$this->web folder)===0)
    $requri=substr($requri,strlen($this->web folder));
  $this->request uri parts = $requri ? explode('/',$requri) : array();
  return $this;
}
```

在 explode_http_request 方法中，通过$_SERVER 变量获取了用户请求的 URI 信息。由于框架的根目录不一定是服务器的根目录，我们可以通过比对该 URI 和框架根目录来调校该 URI 信息，将 URI 信息中的框架根目录去掉。若调校后的 URI 信息长度为 0，则将requst_uri_parts 属性的值定义为一个空数组。否则，则使用 explode()函数分离 URI 信息中通过“/”连接的各项信息，然后将这些信息以数组的形式存入 request_uri_parts 属性中。

```
function parse http request() {
  $this->params = array();
  $p = $this->request_uri_parts;

  /*判断数组$p 是否有索引为 0 的元素、该元素值的长度是否为 0、以及该元素值是否不以"?"
  开头，若是，则将该值存入 controller 属性中。*/
  if (isset($p[0]) && $p[0] && $p[0][0]!='?')
    $this->controller=$p[0];

  /*判断数组$p 是否有索引为 1 的元素、该元素值的长度是否为 0、以及该元素值是否不以"?"
  开头，若是，则将该值存入 action 属性中。*/
  if (isset($p[1]) && $p[1] && $p[1][0]!='?')
    $this->action=$p[1];

  /* 判断数组$p 中除了上述两个元素之外是否还有其他元素，若是，则将剩余的元素存入 params
  属性中*/
  if (isset($p[2]))
    $this->params=array slice($p,2);
  return $this;
}
```

在 parse_http_request 方法中，我们把 request_uri_parts 属性的值赋给一个局部变量$p，然后判断变量$p 的头两个元素是否存在、这两个元素的值的长度是否为零、以及它们的首位是否为半角问号。若变量$p 的头两个元素存在、它们的值的长度不为零、且首位不是半角问号，则将第一个元素的值存入 controller 属性中，第二个元素的值存入 action 属性中。

最后，再判断是否还存在其他的元素；若有其他的元素，则将剩余的用户一并存入 params 属性中。

至此，用户请求分析完毕。

在这里还要提一下 Apache 服务器的 URL 改写模块。正是由于我们在 Apache 服务器上启用了 URL 改写模块并在框架目录下配置了分布式配置文件（.htaccess），才实现了将框架目录下的 index.php 页面做为框架入口的目的。而在框架目录下的 index.php 页面中，通过实例化的派生自 KISS_Controller 类的 Controller 类，调用了该实例化对象的 explode_http_request 和 parse_http_request 方法才达到了获取并解析用户请求的目的。

（2）导入控制器文件（route_request）

```
function route_request() {

  //拼接所需 controller 文件所在路径
  $controllerfile=$this->controller_path.$this->controller.'/'.$this->action.
  '.php';

  /*判断所需控制器名称是否包含非法字符，以及上面拼接的路径下是否存在所需的控制器文件，若
  包含非法字符或文件不存在，则调用 request_not_found 方法，返回 404 页面。*/
  if (!preg_match('#^[A-Za-z0-9_-]+$#',$this->controller) || !file_exists
  ($controllerfile))
    $this->request_not_found('Controller file not found: '.$controllerfile);

  //拼接行为函数名称
  $function='_'.$this->action;

  /*判断拼接的行为函数名称是否包含非法字符、以及该函数是否存在，若不存在，调用
  request_not_found 方法，返回 404 页面。*/
  if (!preg_match('#^[A-Za-z_][A-Za-z0-9_-]*$#',$function) || function_exists
  ($function))
    $this->request_not_found('Invalid function name: '.$function);

  //导入所需的控制器文件
  require($controllerfile);

  //判断所需的行为函数是否存在，若不存在，则调用 request_not_found 方法，返回 404 页面
  if (!function_exists($function))
    $this->request_not_found('Function not found: '.$function);

  //通过 call_user_func_array 函数将用户通过 URL 传递的参数代入控制器文件中的行为函
  数中
  call_user_func_array($function,$this->params);
  return $this;
}
```

在 route_request 方法中，使用 controller_path、controller 和 action 属性拼接了控制器文件所在路径，并判断该控制器文件是否存在，控制器文件中是否存在指定的行为函数，若存在则导入该控制器文件，若不存在，则返回 404 页面。

（3）输出 404 页面（request_not_found）

```
function request_not_found($msg='') {
  header("HTTP/1.0 404 Not Found");
  die('<html><head><title>404 Not Found</title></head>
  <body><h1>Not Found</h1><p>'.$msg.'<p>The requested URL was not found on
  this server.</p>
```

```
  <p>Please go <a href="javascript: history.back(1)">back</a> and try again.</p>
  <hr /><p>Powered By: <a href="http://kissmvc.com">KISSMVC</a></p></body>
  </html>');
}
```

该方法只在 route_request 方法中调用。当 route_request 方法没有找到通过 explode_http_request 和 parse_http_request 方法分离出来的控制器和行为时，就会调用 request_not_found 方法，返回 404 页面。

现在，我们再回过头去看看应用程序入口 index.php 文件中创建的基于 Controller 类的名为$controller 的对象。我们发现，在创建这个名为$controller 的对象时，指定了四个参数：控制器目录（APP_PATH.'controllers/'）、框架目录（WEB_FOLDER）、默认控制器（'main'）和默认行为函数（'index'）。

当我们加载 index.php 文件时，就自动加载了“app/controllers/main/”目录下的名为 index.php 的文件中的同名行为函数。该函数通过 call_user_func_array()函数调用执行，并输出结果。这就是为什么我们能看到如图 13-2 所示页面的原因。

如果我们在浏览器中输入 http://www.greatwall.tea/chapter13/users/details/a_florrick，结果会是什么呢？由于并没有定义名为 users 的控制器，名为 details 的方法，能看到的只能是 404 页面（如图 13-3 所示）。页面上提示我们没有定义 users/details.php 文件。

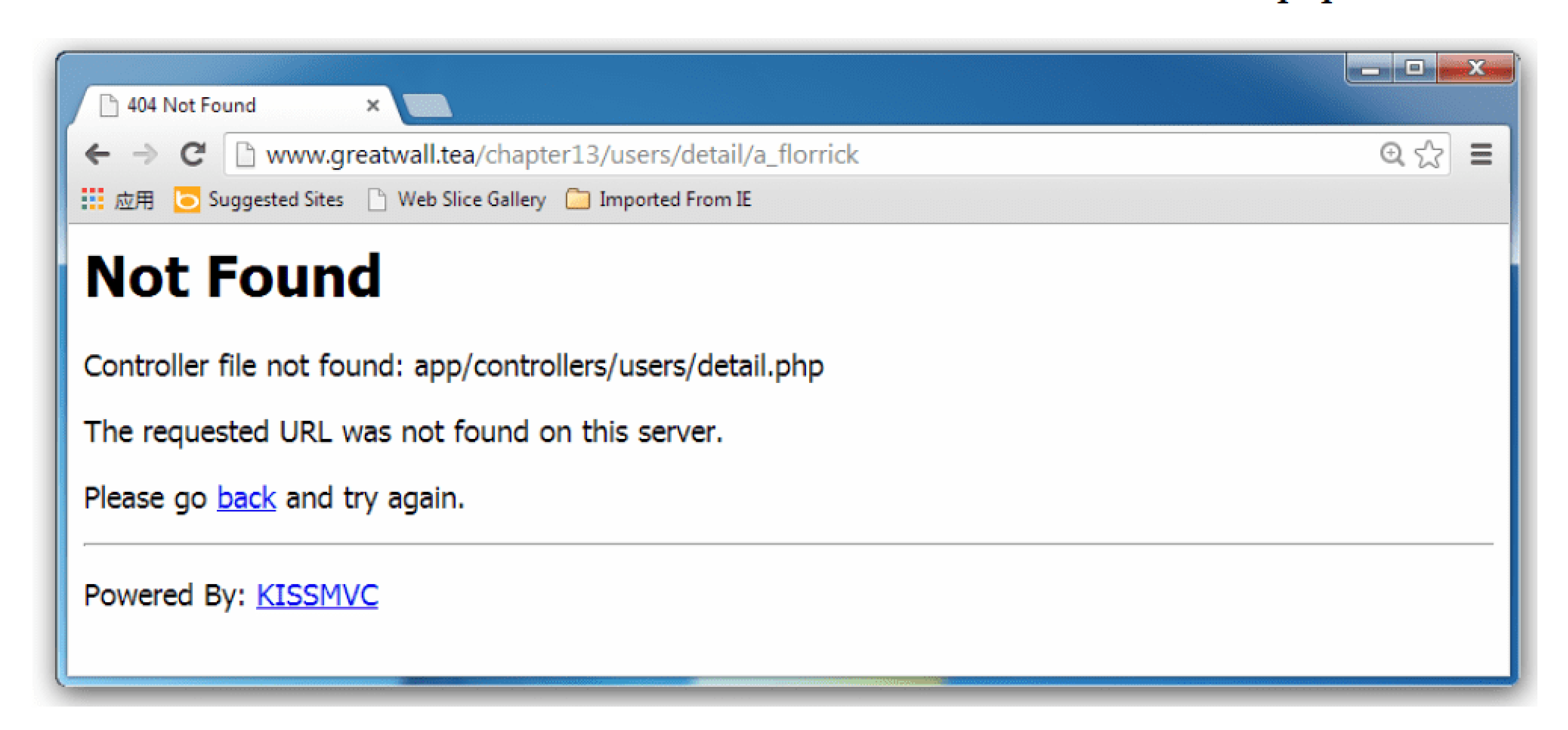


图 13-3　未找到用户请求的方法和属性而返回 404 错误

13.2.4　视图（KISS_View）

在上一小节中提到，之所以可以看到如图 13-2 所示的页面，是因为我们通过 call_user_func_array()函数调用了“app/controllers/main/”目录下的名为 index.php 的文件中的同名行为函数。现在我们来看看这个函数的内容：

```
<?php
function _index($msg='Hello World!') {
  $view = new View(APP_PATH.'views/layout.php');
  $view->set('msg',$msg);
  $view->dump();
}
?>
```

该函数有一个参数$msg。当用户没有指定参数值时，其默认值为“Hello World!”。在函数中，创建了一个基于 View 类的对象，并将其命名为$view。这个名为$view 的视图对象中引用了“app/views/”目录下名为 layout.php 的视图文件，该文件的内容如下：

```
<html>
<head><title><?php echo $GLOBALS['sitename'];?></title></head>
<body>
<h1><?php echo $msg;?></h1>
</body>
</html>
```

在这个视图文件中，我们除了输出在 index.php 文件中定义的全局变量 sitename 之外，还输出了一个名为$msg 的变量，而这个变量是通过$view 对象的 set 方法设置的。在_index 行为属性的末尾，使用$view 对象的 dump 方法向用户浏览器直接输出了这个页面。

这是怎么实现的呢？为了解答这个问题，我们还是先来看看这个名为 KISS_View 的视图类。

在 KISS_View 类中，定义了两个受保护的属性，它们分别用来存放模板文件路径($file)和模板文件里定义的所有变量的数组（$vars）。

然后我们在 KISS_View 类中定义了九种方法，实现如下两大类功能。

1. 设置属性

在 KISS_View 类中，定义了三种方法来设置受保护的属性，它们分别是 set、__set 和 add。其中，set 方法可以将模板文件中定义的变量以数组元素的形式注册到$vars 属性中；而__set 方法则通过调用 set 方法向$vars 属性中注册任意变量；add 方法则用来向$vars 属性中注册模板文件中定义的数组。

2. 生成页面

在 KISS_View 类中，我们还定义了两组六种方法来生成页面。它们的内容分别如下所示。

（1）页面缓存

在该组中的方法都是将生成的页面缓存在服务器内存中，并返回用于获取缓存页面的句柄。我们可以在执行这些方法后，使用 echo 命令或者 print 命令输出缓存的页面。这些用于页面缓存的方法分别是 fetch、do_fetch 和 do_fetch_str。其中，fetch 用来缓存$var 属性中各元素及其他临时变量与模板页面拼合后的结果，并返回调用该结果的句柄；do_fetch 则用来缓存$var 属性中各元素及其他临时变量与模板页面拼合后的结果，并返回调用该结果的句柄；do_fetch_str 则用指定的字符串覆盖已经缓存的$var 属性中各元素及其他临时变量与模板页面的拼合结果，并返回调用该字符串的句柄。

（2）直接显示

在该组中的方法都是直接将生成的页面传递给用户。这些用于直接显示生成页面的方法分别是 dump、do_dump 和 do_dump_str。其中，dump 用来直接向用户浏览器输出$var 属性中各元素及其他临时变量与模板页面拼合后的结果；do_dump 则用来直接向用户浏览器输出$var 属性中各元素及其他临时变量与指定模板页面拼合后的结果；do_dump_str 则指定的字符串覆盖已经缓存的$var 属性中各元素及其他临时变量与模板页面的拼合结果。

说到底，fetcher 组方法和 dumper 组方法的差别就在于 fetcher 组的方法缓存已生成的页面，并返回调用已缓存页面的句柄，而 dumper 组的方法则直接向用户浏览器输出已生成的页面。这个差别就是通过 ob_start()和 ob_get_clean()函数来实现的。前者用来打开缓存开关，而后者则提供调用缓存数据的句柄。

下面是 KISS_View 类的脚本。

```
abstract class KISS_View {
  protected $file='';
  protected $vars=array();

  //初始化受保护属性$file 和$vars
  function   construct($file='',$vars='')  {
    if ($file)
      $this->file = $file;
    if (is array($vars))
      $this->vars=$vars;
    return $this;
  }

  //定义魔术方法__set，通过调用 set 方法来向$var 属性注册任意变量
  function __set($key,$var) {
    return $this->set($key,$var);
  }

  //定义 set 方法，用来向$var 属性注册模板页面上定义的变量
  function set($key,$var) {
    $this->vars[$key]=$var;
    return $this;
  }

  //定义 add 方法，用来向$var 属性注册模板页面上定义的数组
  function add($key,$var) {
    $this->vars[$key][]=$var;
  }

  /*定义 fetch 方法，用于缓存$var 属性中各元素、临时变量与模板页面拼合后的结果，
  并返回调用该结果的句柄。 */
  function fetch($vars='') {
    if (is array($vars))
      $this->vars=array merge($this->vars,$vars);
    extract($this->vars);
    ob start();
    require($this->file);
    return ob get clean();
  }

  //定义 dump 方法，用于直接向用户浏览器输出$var 属性中各元素、临时变量与模板页面拼合
  后的结果
  function dump($vars='') {
    if (is array($vars))
      $this->vars=array merge($this->vars,$vars);
    extract($this->vars);
    require($this->file);
  }

  /*定义 do_fetch 方法，用于缓存$var 属性中各元素、临时变量与指定模板页面拼合后的结果，
  并返回调用该结果的句柄。 */
```

```
  static function do_fetch($file='',$vars='') {
    if (is_array($vars))
      extract($vars);
    ob_start();
    require($file);
    return ob_get_clean();
  }

  /*定义 do_dump 方法，用于直接向用户浏览器输出$var 属性中各元素、临时变量
  与指定模板页面拼合后的结果。 */
  static function do_dump($file='',$vars='') {
    if (is_array($vars))
      extract($vars);
    require($file);
  }

  /* 定义 do_fetch_str 方法，可用指定的字符串覆盖将已缓存的$var 属性中各元素、临时
  变量与指定模板页面拼合结果。*/
  static function do_fetch_str($str,$vars='') {
    if (is_array($vars))
      extract($vars);
    ob_start();
    eval('?>'.$str);
    return ob_get_clean();
  }

  /* 定义 do_dump_str 方法，直接向用户浏览器输出用于覆盖已缓存的$var 属性中各元素、
  临时变量与指定模板页面拼合结果的字符串。*/
  static function do_dump_str($str,$vars='') {
    if (is_array($vars))
      extract($vars);
    eval('?>'.$str);
  }
}
```

在了解了 KISS_View 类的脚本后，再回过头去看看“app/controllers/main/”目录下的名为 index.php 的文件中的同名行为函数及其调用的模板文件。这个名为_index()的函数带有一个参数$msg，其默认值为“Hello World”。在_index()里，我们实例化了派生自 KISS_View 类的 View 类，然后调用了该对象的 set 方法将参数$msg 的值以“msg”为索引存入了对象的$vars 属性中。接着调用该对象的 dump 方法向用户浏览器输出页面。在 dump 方法中，我们将$vars 属性中的元素通过 extract()函数分离出来，然后再导入模板文件，从而实现$vars 属性中的各元素与模板文件的拼合过程。

13.2.5　模型（KISS_Model）

就像我们在 13.1.1 小节里提到的那样：“在实际的应用程序开发中，并不是所有的用户请求都需要调用模型或视图的。有的用户请求涉及到的模型可能非常复杂，一个模型解决不了问题，这时就需要调用多个模型，而其中肯定会有些模型做为中间件而不需要调用视图。”

KISSMVC 框架的欢迎页面就没有用到模型。用户将请求发送到框架，该请求被 Apache 服务器按照分布式配置文件中定义的规则转发到了框架的唯一入口。在框架获取并解析了用户请求之后，通过调用所需的视图类来向用户浏览器返回用户请求的数据。

这一过程中，我们并没有用到模型。那么模型到底在什么情况下才会使用呢？

一般来说，涉及到与数据库的交互时，才会用到模型。所谓模型，其实就是关系型数据库中 CRUD 操作的对象化。

在 KISSMVC 框架中，KISS_Model 类是三个核心抽象类中最复杂的一个，但是它并不难懂。在这一小节里，我们就来分析一下这个模型类。

在这个模型类中一共定义了六个受保护的属性，它们分别是：主键名（$pkname）、表名（$tablename）、数据库句柄函数名（$dbhsfnname）、引号风格（$QUOTE_STYLE）、数组压缩（$COMPRESS_ARRAY）和记录集（$rs）。

该类的属性和初始化方法如下：

```
abstract class KISS_Model  {

  protected $pkname;
  protected $tablename;
  protected $dbhfnname;
  protected $QUOTE_STYLE='MYSQL'; //三种可选引号样式： MYSQL,MSSQL,ANSI
  protected $COMPRESS_ARRAY=true;
  public $rs = array(); //存放所有对象属性变量

  function __construct($pkname='',$tablename='',$dbhfnname='getdbh',$quote_
  style='MYSQL',
                       $compress_array=true) {
    $this->pkname=$pkname;            //编号自增长主键名
    $this->tablename=$tablename;      //数据库表名
    $this->dbhfnname=$dbhfnname;      //数据库句柄函数名
    $this->QUOTE_STYLE=$quote_style; //拼接 SQL 语句时，各字段名前后使用的引号风格
    $this->COMPRESS_ARRAY=$compress_array; //绑定值到 SQL 语句中各字段名时，某
    值为数组，是否压缩
  }
  //以下为模型类的各种方法，暂略
  …
}
```

除此之外，模型类中还定义四种基础 CRUD 方法、四种高级查询方法和七种辅助方法，共计 15 个方法。

现在我们就来看看这些方法。

1．基础CRUD方法

我们之前说过，模型类的目的是与数据库交互，完成数据的存储和查询任务，而这些任务就会涉及到 CRUD 操作。KISSMVC 框架的模型类一共有四个基础的 CRUD 方法，它们分别是 create、retrieve、update 和 delete，分别用来向数据库添加一条记录、从数据库中提取一条记录、更新数据库中的一条指定的记录以及把一条指定记录从数据库中删除。

编写这四个基础 CRUD 方法的思路是，先通过辅助方法（如 set）将需要添加、查询、更新或删除的数据记录的字段名和值存入对象私有属性$rs 中，然后通过遍历$rs 中的各元素来拼接 SQL 语句，最后执行拼接好的 SQL 语句，并返回对象。

下面，我们先来看看 create 方法的脚本：

```
//以下方法用于向数据库中添加一条记录
function create() {
  //获取数据库操作句柄
  $dbh=$this->getdbh();

  //将主键名赋值给局部变量$pkname
  $pkname=$this->pkname;

  //定义两个空字符串用来拼接查询语句
  $s1=$s2='';

  /*将私有属性$rs 中的各元素的索引用半脚逗号连接存入局部变量$s1，
  然后将对应数量的"?"用半脚逗号连接存入局部变量$s2 中 */
  foreach ($this->rs as $k => $v)
    if ($k!=$pkname || $v) {
      $s1 .= ','.$this->enquote($k);
      $s2 .= ',?';
    }
  //拼接 SQL 语句
  $sql = 'INSERT INTO '.$this->enquote($this->tablename).' ('.substr($s1,1).')
  VALUES ('.substr($s2,1).')';

  //准备执行 SQL 语句
  $stmt = $dbh->prepare($sql);

  //将属性$rs 中各元素的值绑定到 SQL 语句中定义的各字段名
  $i=0;
  foreach ($this->rs as $k => $v)
    if ($k!=$pkname || $v)
      $stmt->bindValue(++$i,is_scalar($v) ?
                          $v : ($this->COMPRESS_ARRAY ? gzdeflate(serialize
                          ($v)) : serialize($v)) );

  //执行准备好的 SQL 语句
  $stmt->execute();

  /* 通过 rowCount()函数获取执行此 SQL 语句后受到影响的记录数量，若该函数返回 FALSE，
  则添加失败。
  此时，直接返回 FALSE，退出该方法。 */
  if (!$stmt->rowCount())
    return false;

  //获取已添加记录的序号，并将其存入属性$pkname 中
  $this->set($pkname,$dbh->lastInsertId());

  //返回对象
  return $this;
}
```

在 create 方法的脚本中，我们就是按照上文中提到的思路编写的。需要注意的是，有的时候数据库中的某个字段的值可能会是一个非标准变量，这时就要通过合理使用 serialize()和 unserialize()函数来简化数据库的表结构。在上面的脚本中，我们在绑定属性$rs 中各元素的值到 SQL 语句中各字段名时，根据属性$COMPRESS_ARRAY 的值来判断是否压缩非标变量序列化后的值。这样一来，数据库中各字段里也可以存储数组、对象之类的非标变量值了。

下面再来看看如何查询数据库中符合条件的数据记录。

```
//以下方法用于在数据库中指定主键值的一条数据记录，参数为主键值
function retrieve($pkvalue) {
  //获取数据库操作句柄
  $dbh=$this->getdbh();

  //拼接 SQL 语句
  $sql = 'SELECT * FROM '.$this->enquote($this->tablename).' WHERE '.$this->
  enquote($this->pkname).'=?';

  //准备执行 SQL 语句
  $stmt = $dbh->prepare($sql);

  //绑定主键值到 SQL 语句中指定的主键名
  $stmt->bindValue(1,(int)$pkvalue);

  //执行 SQL 语句
  $stmt->execute();

  //将 SQL 语句执行结果以数组的形式存入局部变量$rs 中，各元素的索引为其值对应的字段名
  $rs = $stmt->fetch(PDO::FETCH_ASSOC);

  /*如果局部变量$rs 不为空，则遍历该数组，以便还原之前被压缩的非标变量值，
  并将它们存入属性$rs 中的对应元素里。*/
  if ($rs)
    foreach ($rs as $key => $val)
      if (isset($this->rs[$key]))
        $this->rs[$key] = is_scalar($this->rs[$key]) ?
         $val : unserialize($this->COMPRESS_ARRAY ? gzinflate($val) : $val);

    //返回对象
    return $this;
}
```

在 retrieve 方法的脚本中，我们定义了如何从数据库中获取指定主键值的数据记录。获取此类数据记录的 SQL 语句较为简单，拼接 SQL 语句时也很轻松，只要注意 WHERE 子句的内容就好。在将查询到的数据存入属性$rs 中之后，我们遍历数组，还原可能被压缩或序列化的非标变量值。最后，返回对象，以便调用对象的其他方法。

下面两个方法与 create 和 retrieve 方法类似，大家可以根据注释自己揣摩，我们就不再赘述了。

```
//以下方法用于更新数据库中指定主键值的一条数据记录
function update() {
  //获取数据库操作句柄
  $dbh=$this->getdbh();

  //定义局部变量$s，用于拼接 SQL 语句。这一点与 create 方法中的局部变量$s1 和$s2 类似
  $s='';
  foreach ($this->rs as $k => $v)
    $s .= ','.$this->enquote($k).'=?';
  $s = substr($s,1);
  $sql = 'UPDATE '.$this->enquote($this->tablename).
         ' SET '.$s.' WHERE '.$this->enquote($this->pkname).'=?';
```

```
  //准备执行 SQL 语句
  $stmt = $dbh->prepare($sql);

  //绑定 UPDATE 主句中的字段名和值
  $i=0;
  foreach ($this->rs as $k => $v)
    $stmt->bindValue(++$i,is scalar($v) ?
                      $v : ($this->COMPRESS ARRAY ? gzdeflate(serialize($v)) :
                      serialize($v)) );

  //绑定 WHERE 子句中的字段名和值
  $stmt->bindValue(++$i,$this->rs[$this->pkname]);

  //执行 SQL 语句，并返回执行结果
  return $stmt->execute();
}

//以下方法用于删除数据库中指定主键值的一条数据记录
function delete() {
  //获取数据库操作句柄
  $dbh=$this->getdbh();

  //拼接 SQL 语句
  $sql = 'DELETE FROM '.$this->enquote($this->tablename).' WHERE '.$this->
  enquote($this->pkname).'=?';

  //准备执行 SQL 语句
  $stmt = $dbh->prepare($sql);

  //绑定 WHERE 子句中的字段名和值
  $stmt->bindValue(1,$this->rs[$this->pkname]);

  //执行 SQL 语句，并返回执行结果
  return $stmt->execute();
}
```

2．高级查询方法

在基础 CRUD 方法中的查询方法（retrieve）固然很妙，但是却只能根据指定的主键值来查询一条数据记录，实在不能完全满足我们的查询需求。这时可以使用模型类中定义的四种高级查询方法来丰富我们的查询方式，它们分别是 exists、retrieve_one、retrieve_many 和 select，分别用来判断某条数据记录是否存在、查询一条满足指定条件的数据记录、查询多条满足条件的数据记录并返回这些记录的所有字段和查询多条满足条件的数据记录并返回这些记录的指定字段。

下面我们就来看看这四种高级查询方法的脚本：

```
/* 以下方法用来查询某条数据是否存在，若存在，则根据参数$checkdb 的值返回 TRUE 或 1；
若不存在，则返回 FALSE。*/
function exists($checkdb=false) {
  if ((int)$this->rs[$this->pkname] < 1)
    return false;
  if (!$checkdb)
    return true;

  //获取数据库操作句柄
```

```
  $dbh=$this->getdbh();

  //拼接 SQL 语句
  $sql = 'SELECT 1 FROM '.$this->enquote($this->tablename).
         ' WHERE '.$this->enquote($this->pkname)."='".$this->rs[$this->
         pkname]."'";

  //执行 SQL 语句，并将执行结果存入局部变量$result 中
  $result = $dbh->query($sql)->fetchAll();

  //返回$result 中记录的数量
  return count($result);
}

/* 以下方法用于查询数据库一条满足指定条件的数据记录。若存在多条满足此指定条件的记录，
则返回第一条记录。*/
function retrieve_one($wherewhat='',$bindings='') {

  //获取数据库操作句柄
  $dbh=$this->getdbh();

  //判断参数$bindings 是否为标准变量。若是，则将其以数组的形式存入局部变量$bindings 中
  if (is_scalar($bindings))
    $bindings= trim($bindings) ? array($bindings) : array();

  //拼接 SQL 语句
  $sql = 'SELECT * FROM '.$this->enquote($this->tablename);

  //判断参数$wherewhat 是否设置。若设置，则将其添加到 SQL 语句末尾
  if ($wherewhat)
    $sql .= ' WHERE '.$wherewhat;

  //在 SQL 语句末尾添加 LIMIT 子句，以便在多条记录满足指定条件时只输出第一条满足条件的记录
  $sql .= ' LIMIT 1';

  //准备执行 SQL 语句
  $stmt = $dbh->prepare($sql);

  //读取局部变量$bindings 的值，并将它们与 WHERE 子句中指定的各字段名绑定
  $i=0;
  foreach($bindings as $v)
    $stmt->bindValue(++$i,$v);

  //执行 SQL 语句
  $stmt->execute();

  //将查询结果存入局部变量$rs 中
  $rs = $stmt->fetch(PDO::FETCH_ASSOC);

  /* 如果局部变量$rs 不为空，则遍历该数组，以便还原之前被压缩的非标变量值，
  并将它们存入属性$rs 中的对应元素里。*/
  if (!$rs)
    return false;
  foreach ($rs as $key => $val)
    if (isset($this->rs[$key]))
      $this->rs[$key] = is_scalar($this->rs[$key]) ?
                        $val : unserialize($this->COMPRESS_ARRAY ? gzinflate
                        ($val) : $val);
```

```
  //返回对象
  return $this;
}

  //以下方法用于查询满足指定条件的所有记录。与 retrieve_one 方法唯一不同的是，
  retrieve_one 方法在拼接 SQL 语句时，添加了 LIMIT 子句，而 retrieve_many 方法则没有。*/
  function retrieve_many($wherewhat='',$bindings='') {
  $dbh=$this->getdbh();
  if (is_scalar($bindings))
    $bindings=trim($bindings) ? array($bindings) : array();
  $sql = 'SELECT * FROM '.$this->tablename;
  if ($wherewhat)
    $sql .= ' WHERE '.$wherewhat;
  $stmt = $dbh->prepare($sql);
  $i=0;
  foreach($bindings as $v)
    $stmt->bindValue(++$i,$v);
  $stmt->execute();
  $arr=array();
  $class=get_class($this);
  while ($rs = $stmt->fetch(PDO::FETCH_ASSOC)) {
    $myclass = new $class();
    foreach ($rs as $key => $val)
      if (isset($myclass->rs[$key]))
        $myclass->rs[$key] = is_scalar($myclass->rs[$key]) ?
                             $val : unserialize($this->COMPRESS_ARRAY ?
                             gzinflate ($val) : $val);
    $arr[]=$myclass;
  }
  return $arr;
}

/* 以下方法用于查询满足指定条件的所有记录（指定字段）。该方法与 retrieve_many 类似，
只是在拼接 SQL 语句和绑定 SQL 语句中的字段名和值时，加入了参数$selectwhat 中指定的内容。*/
function   select($selectwhat='*',$wherewhat='',$bindings='',$pdo_fetch_mode
=PDO::FETCH_ASSOC) {
  $dbh=$this->getdbh();
  if (is_scalar($bindings))
    $bindings=trim($bindings) ? array($bindings) : array();
  $sql = 'SELECT '.$selectwhat.' FROM '.$this->tablename;
  if ($wherewhat)
    $sql .= ' WHERE '.$wherewhat;
  $stmt = $dbh->prepare($sql);
  $i=0;
  foreach($bindings as $v)
    $stmt->bindValue(++$i,$v);
  $stmt->execute();
  return $stmt->fetchAll($pdo_fetch_mode);
}
```

3．辅助方法

除开上述八种主要的 CRUD 操作方法之外，KISSMVC 框架的模型类还为我们提供了七种辅助方法，用于设置和获取某些属性的值。它们是 set、get、__set、__get、getdbh、enquote 和 merge，分别用来设置属性$rs 元素、获取属性$rs 中指定元素的值、获取数据库操作句柄、定义 SQL 语句引号风格和合并数组到$rs 属性中。

下面我们就来看看这七种方法的脚本：

```
//以下方法用于获取$rs 属性中指定元素的值
function get($key) {
  return $this->rs[$key];
}

//以下方法用于设置$rs 属性中指定元素的值
function set($key, $val) {
  if (isset($this->rs[$key]))
    $this->rs[$key] = $val;
  return $this;
}

//同 get 方法
function __get($key) {
  return $this->get($key);
}

//同 set 方法
function __set($key, $val) {
  return $this->set($key,$val);
}

//以下方法通过 call_user_func()函数调用用户自定义的数据库连接函数
protected function getdbh() {
  return call_user_func($this->dbhfnname);
}

//以下方法用于定义在拼接 SQL 语句中，使用的引号风格
protected function enquote($name) {
  if ($this->QUOTE_STYLE=='MYSQL')
    return '`'.$name.'`';
  elseif ($this->QUOTE_STYLE=='MSSQL')
    return '['.$name.']';
  else
    return '"'.$name.'"';
}

//以下方法用于向$rs 属性添加元素
function merge($arr) {
  if (!is_array($arr))
    return $this;
  foreach ($arr as $key => $val)
    if (isset($this->rs[$key]))
      $this->rs[$key] = $val;
  return $this;
}
```

13.2.6　使用控制器操控模型和视图

在上一小节里，我们提到，在模型类的辅助方法 getdbh 中，我们使用了 call_user_func()函数获取用户自定义的获取数据库操作句柄的函数。这样就赋予了给我们更大的选择数据库类型的权利。如果读者不想自己定义获取数据库连接的函数，KISSMVC 框架也为我们写好了一个函数，直接修改一下就可以用了。

打开 KISSMVC 框架目录下的 index.php 文件，找到如下内容：

```
//===============================================
//Database
```

```
//==========================================
/*
function getdbh() {
  if (!isset($GLOBALS['dbh']))
    try {
      $GLOBALS['dbh'] = new PDO('sqlite:'.APP_PATH.'db/kissmvc.sqlite');
      //$GLOBALS['dbh'] = new PDO('mysql:host=localhost;dbname=kissmvc',
'username', 'password');

    } catch (PDOException $e) {
      die('Connection failed: '.$e->getMessage());
    }
  return $GLOBALS['dbh'];
}
*/
```

如果需要启用该 getdbh()函数，请移除该函数前后的注释符（/* …*/）即可。若需要使用 SQLite 数据库，则需要注明数据库文件的路径。若需要使用 MySQL 数据库，则需要注释掉 SQLite 数据库的那一行，并将 MySQL 数据库对应的那一行前面的注释符去掉，然后修改主机名、数据库名、用户名和密码。在这里提一句，下面的例子中使用的是 MySQL 数据库，主机名为 localhost，数据库名为 kissmvc，用户名为 kissmvc，密码为 Passw0rd。

随后，我们还需要启动自动加载类文件功能。在 KISSMVC 框架目录下的 index.php 文件中，找到如下脚本：

```
//==========================================
//Autoloading for Business Classes
//==========================================
//Assumes Model Classes start with capital letters and Helpers start with
lower case letters
/*
function __autoload($classname) {
  $a=$classname[0];
  if ($a >= 'A' && $a <='Z')
    require_once(APP_PATH.'models/'.$classname.'.php');
  else
    require_once(APP_PATH.'helpers/'.$classname.'.php');
}
*/
```

移除__autoload()函数前后的注释符（/* …*/）即可。需要注意的是，如果启用自动加载类文件功能，可以免去我们在定义模型、控制器和视图时再使用 require()和 include()函数。但是，模型类的命名必须是大写字母开头，而框架助手类的命名必须是小写字母开头。否则，框架会找不到模型和框架助手而出错。

现在做好了准备工作，来一起看看怎么使用控制器操控模型吧。

假设 kissmvc 数据库中有一张名为 users 的表，表中有如下几个字段：uid、username、fullname 、gender 和 password。大家可以自行在 MySQL 数据库中创建这样一个数据表。

然后进入 kissmvc 框架目录，在应用目录（app）下的 model 文件夹里创建一个名为 user.php 的文件。打开该文件，然后输入如下脚本：

```
<?php
    class User extends Model {
     function User() {
         parent::__construct('uid','user','getdbh');
             $this->rs['uid'] = '';
             $this->rs['username'] = '';
```

```
            $this->rs['password'] = '';
            $this->rs['fullname'] = '';
            $this->rs['gender'] = '';
            $this->rs['created_dt'] = '';
    }
   }
?>
```

这段脚本定义了派生自 Model 类的 User 类。这就意味着，一个模型对应一张数据表。由于我们需要使用自动加载类文件的功能，所以在定义类时，注意首字母大写，这样就可以在使用类时不引用相应的类文件。

在这段脚本中，首先初始化了 User 类的父类 Model，指定了数据表的主键名、表名和数据库操控句柄函数名。其中，parent 关键字指代当前类（User）的父类（Model）；接着定义了 User 类的$rs 属性，将数据库表各字段的名称做为数组元素索引存入$rs 属性中。这样我们就定义好了一个模型。

接下来，我们在应用目录下的 controllers 子文件夹下，新建一个名为 user 的子文件夹；然后在其中新建一个名为 add.php 的文件。打开该文件，输入如下脚本：

```
<?php
   function  add($action=''){
    $msg = '';
    $view = new View(APP_PATH.'views/adduser.php');
        if ($action == 'do'){
            //获取用户提交的数据。
            $username = $_POST['username'];
            $fullname = $_POST['fullname'];
            $gender = $_POST['gender'];
            $password = $_POST['password'];
            $confirm = $_POST['confirm'];

            //验证用户提交的数据。
            if(strlen($username) < 1 || strlen($fullname) < 1 ||
               strpos($fullname, ' ') == FALSE || strlen($password) < 10 ||
               $password <> $confirm) {
                 $msg = "Please ensure the correctness of your input.";
            } else {
                //若用户提交的数据符合要求，则将它们存入数据库。
                $user = new User();
                $user->set('username',$_POST['username']);
                $user->set('fullname',$_POST['fullname']);
                $user->set('gender',$_POST['gender']);
                $user->set('password',$_POST['password']);
                $user->set('created_dt',date('Y-m-d'));
                $user->create();
                $uid = (string) $user->get('uid');
                if($uid){
                    $msg = "This user has been added. The user ID is {$uid}.";

                } else {
                    $msg = "Failed to add this user.";
                }
            }
        }
        $view->set('msg', $msg);
        $view->dump();
   }
?>
```

在这段脚本中，定义了一个名为_add()的函数。该函数与 add.php 同名，会被引用到 KISSMVC 框架目录下 index.php 文件中创建的$controller 对象中。在_add()函数中，实例化了 View 类，引用了位于应用目录下 views 子目录中的名为 adduser.php 模板文件。然后获取并分析用户提交的数组。若用户提交的数据符合要求，则将用户提交的数据绑定到$user 对象的$rs 属性中，然后再使用$user 对象的 create 方法将绑定好的数据存入数据库。随后，再使用$user 对象的 get 方法获取新添加记录的 uid 字段的值，输出给用户。

最后，将生成的$msg 信息绑定到视图文件中的$msg 变量，输出模板页面到用户浏览器。

现在，我们在浏览器中访问 KISSMVC 框架，在地址栏中的地址末尾加上“user/add”，然后回车，会看到如图 13-4 所示的页面。

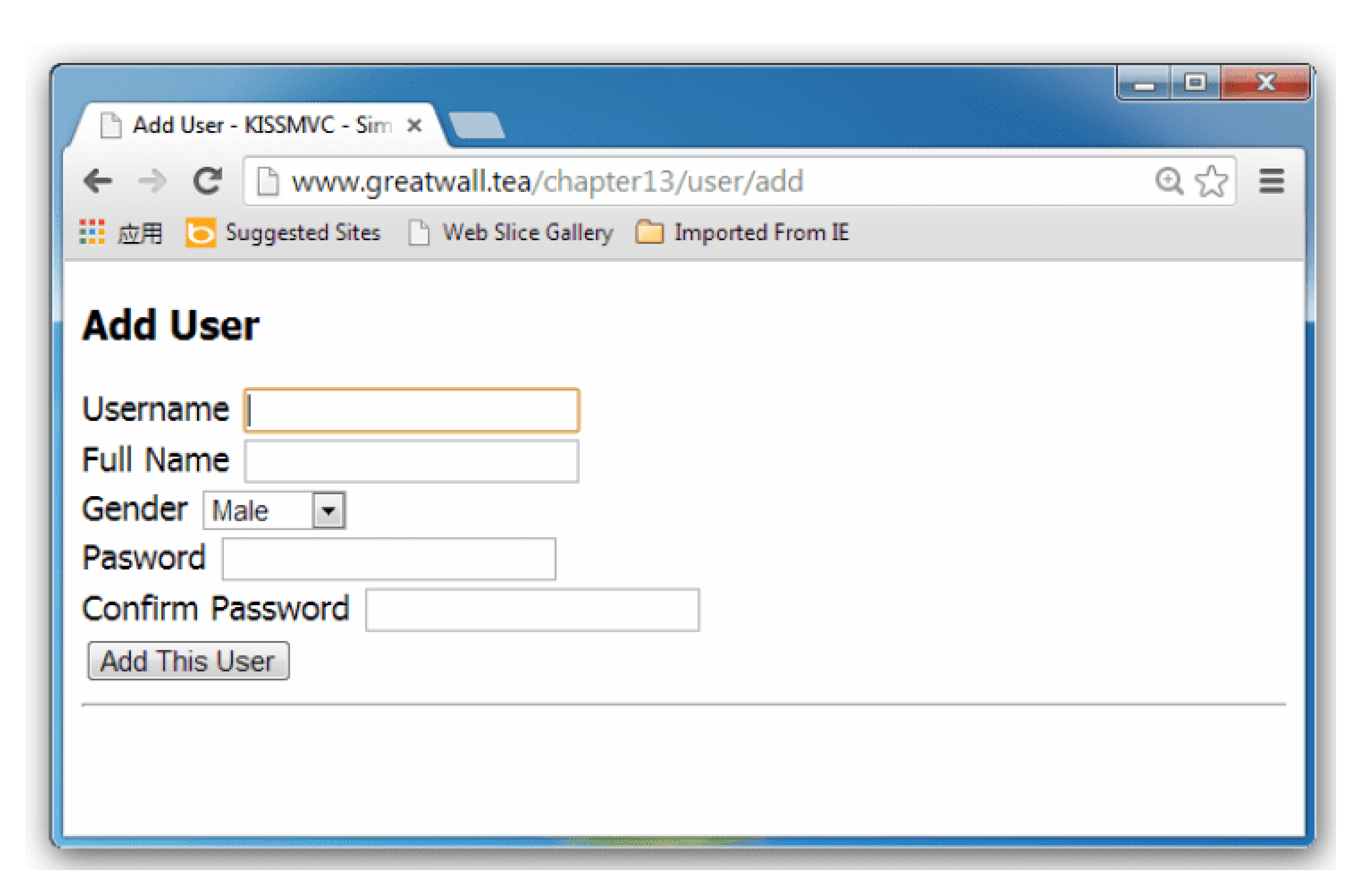


图 13-4　添加用户

当我们填写完表单之后，单击“Add This User”按钮添加该用户。如果数据库连接正常且填写的信息符合要求，会看到如图 13-5 所示的页面。

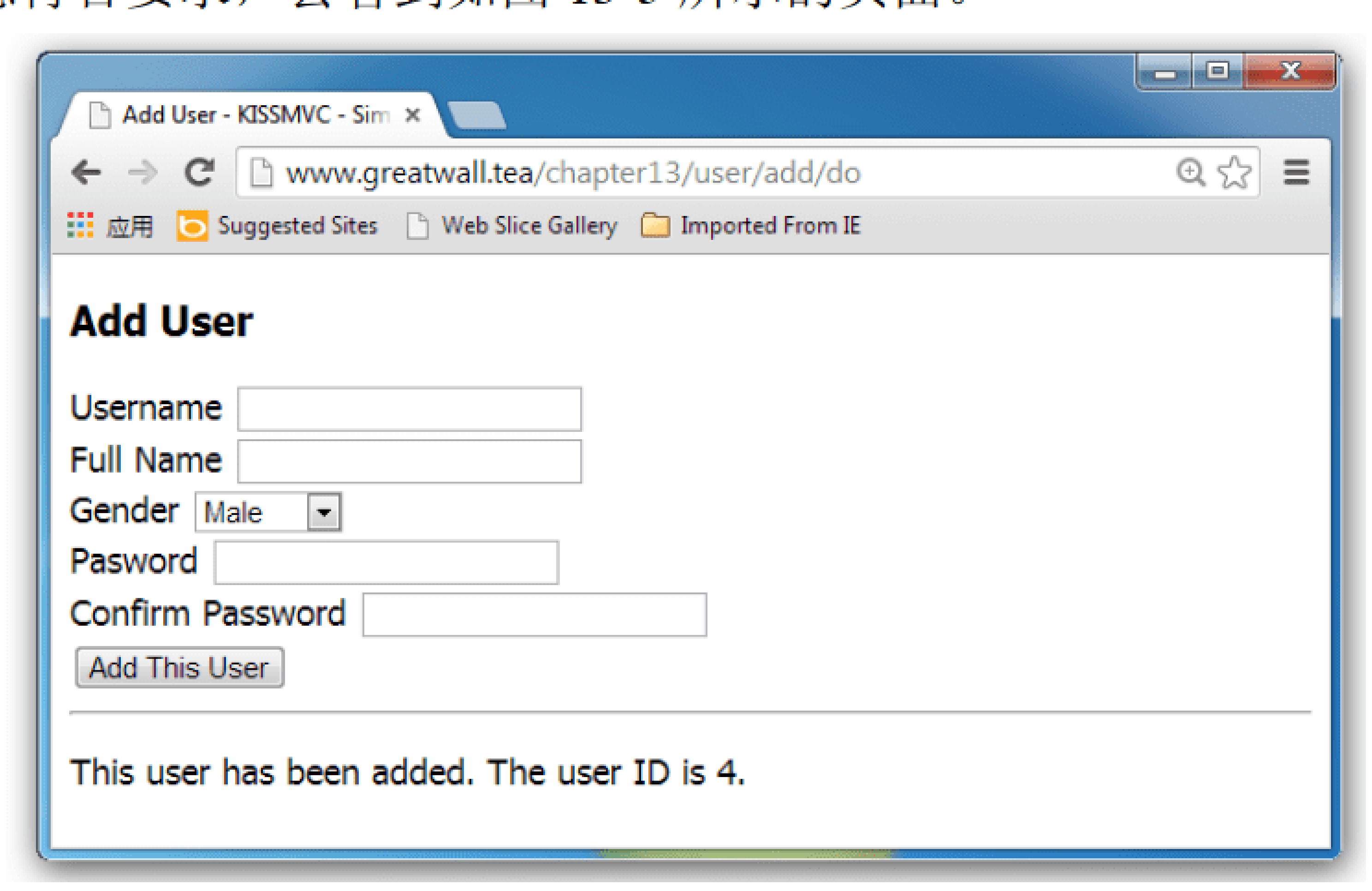


图 13-5　添加用户成功

这样看来，我们成功地使用控制器操控模型完成了向数据库添加数据的任务。

看到这里，有的同学可能会说，这么写程序累不累啊，又是控制器，又是模型的，结果完成的还是一个很简单的任务。我只能说：“兄弟，你太不淡定了！”

还记得我们在本章的开头说过为什么要使用 MVC 框架吗？其中最重要的一句就是“由于这三个部分（指控制器、模型和视图）相互独立，我们对其中某个部件的修改完全不会对其他两个部件造成影响。”为了验证这个说法，现在就在应用目录下的 views 文件夹下再新建一个名为 adduser-bootstrap.php 的文件，使用 Bootstrap UI 框架来重写“添加用户”的页面。然后打开 controllers/user 目录下的 add.php 文件，找到如下内容：

```
$view = new View(APP_PATH.'views/adduser.php');
```

将其替换成：

```
$view = new View(APP_PATH.'views/adduser-admin.php');
```

然后再加载一下这个页面，会发现，这个页面的布局变成了如图 13-6 所示的样子。

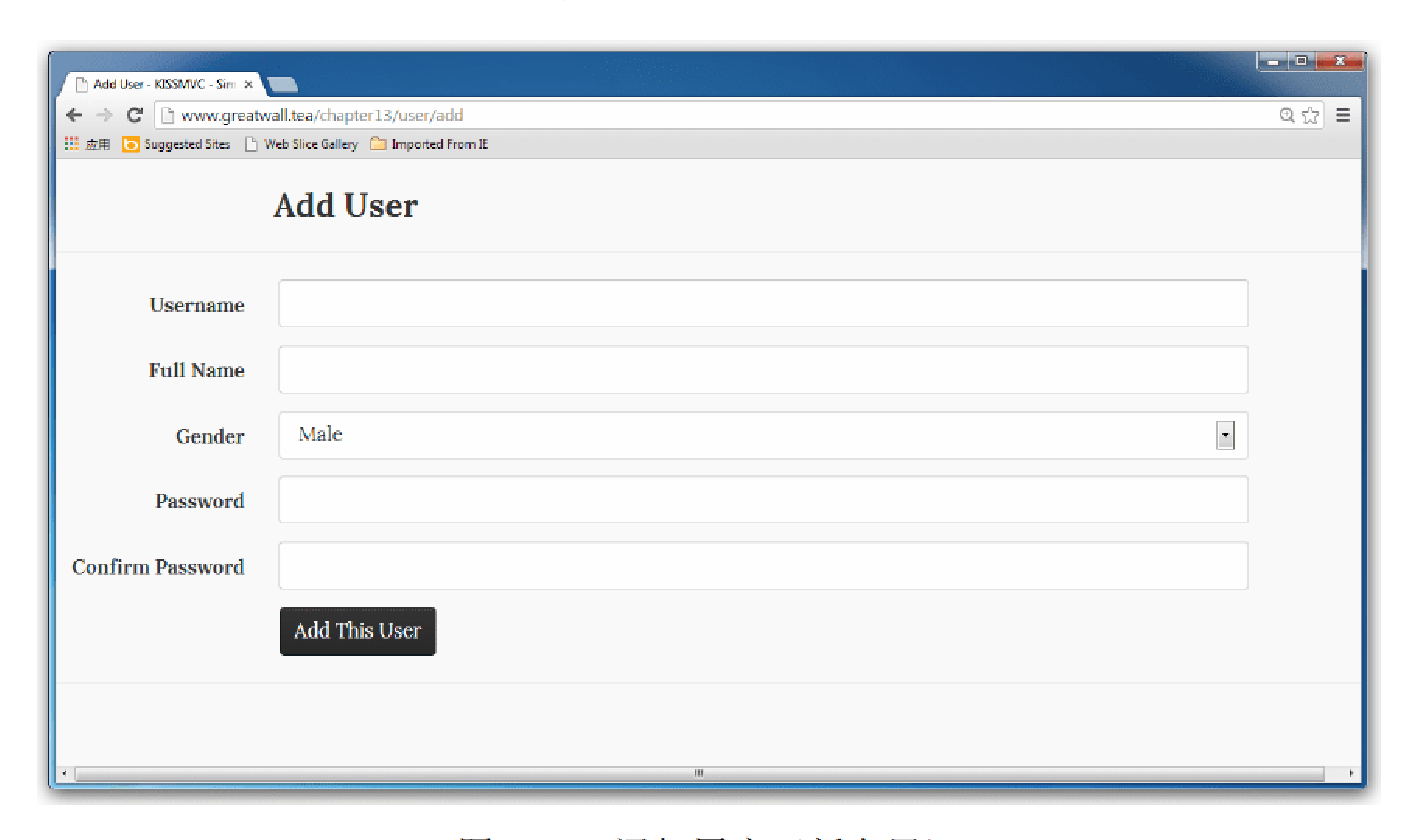


图 13-6　添加用户（新布局）

而当用户添加成功后的页面则如图 13-7 所示。

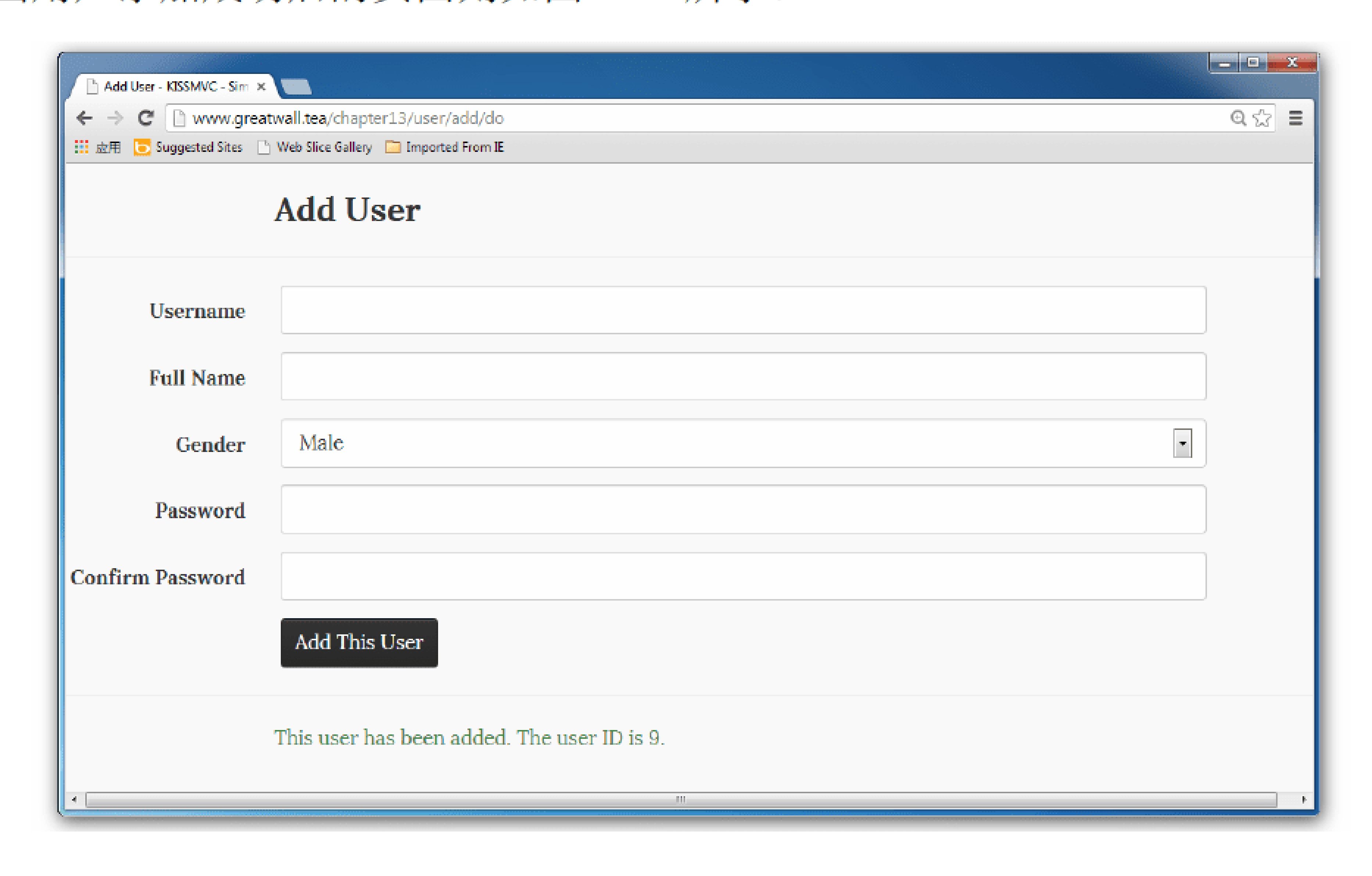


图 13-7　添加用户成功（新布局）

这里，我们只对模板文件进行了修改。为了说明模板文件是相互独立的，在应用目录

的 views 文件夹下新建了一个名为 adduser-bootstrap.php 的模板文件，然后修改了 controllers/user 文件夹下的 add.php 文件。读者完全可以把 adduser-bootstrap.php 的内容复制到 adduser.php 模板文件中。这样一来，完全不用修改 controllers/user 文件夹下的 add.php 文件，就可以实现更换页面布局的目的。

使用 MVC 框架编写的脚本维护起来真是不太方便啊，你觉得呢？

13.3　扩充框架：基于 MVC 的记账工具

还记得我们在第 8～10 章里编写的记账工具吗？那个记账工具虽然是基于对象编程的产物，但却没有使用任何的设计模式，所有的属性和方法都写到了一个类里。这样不利于后续对脚本进行维护，同时也很难对工具的用户界面进行重新设计。

在本小节里，将使用 KISSMVC 框架来重新编写我们的记账工具，将记账工具的业务逻辑、数据库读写和用户界面呈现分离开来。同时，将会为这个记账工具添加一些新的功能，并通过对用户更加友好的界面元素加以呈现。在记账工具的模板部分，我们将使用基于 Bootstrap UI 框架的一套模板[iii]。大家也可以了解一下如何使用 Bootstrap 这一当前十分流行的用户界面框架。

13.3.1　数据规划

按照第 8 章规划的功能，将在本小节里完成对基于 MVC 的记账工具的数据规划工作。表 13-1 规划了项目名称、数据库类型、数据库连接数据和主要功能。

表 13-1　基本数据规划

项　　目	数　　据	
项目名称	PennyCounts – Every Penny Counts!	
项目架构	KISSMVC	
数据库类型	MySQL	
数据库连接	主机名	localhost
	数据库名	kissmvc
	用户名	kissmvc
	密码	Passw0rd*
主要功能	（1）用户登录、注销与验证 （2）逐条/批量添加收入记录 （3）逐条/批量添加支出记录 （4）输出收入/支出列表 （5）修改指定的收入和支出记录 （6）删除指定的收入和支出记录 （7）输出分月统计报表	

[iii] 你可以从 Bootstrap 的官方网站（http://getbootstrap.com）上获取源代码。从 Bootswatch 网站（http://bootswatch.com）上获取开源的 Bootstrap UI 框架主题，关于如何使用 Bootstrap UI 框架，请参考 Bootstrap 官方网站上的文档说明。本书中使用的 Bootstrap UI 框架主题是 Bootswatch 网站上提供的名为 Readable 主题（http://bootswatch.com/readable/）。

在定义了基本信息之后，需要在数据库中建立若干张数据表，用于存放用户、收入记录和消费记录数据。具体规划如表 13-2 所示。

表 13-2　数据表规划

项　　目	字　段　名	数 据 类 型	备　　注
用户表（user）	uid	INT (11)	主键、非空、自增长
	username	VARCHAR (25)	可为空
	fullname	VARCHAR (60)	可为空
	gender	VARCHAR (10)	可为空
	password	VARCHAR (25)	可为空
	avatar	VARCHAR (255)	可为空
	created_dt	DATE	可为空
收入记录表（income）	incid	INT (11)	主键、非空、自增长
	description	VARCHAR (128)	可为空
	amount	FLOAT (8, 2)	可为空
	add_dt	DATE	可为空
	mod_dt	DATE	可为空
	category	VARCHAR (2)	可为空
	uid	INT (11)	外键
支出记录表（expense）	expid	INT (11)	主键、非空、自增长
	description	VARCHAR (128)	可为空
	amount	FLOAT (8, 2)	可为空
	add_dt	DATE	可为空
	mod_dt	DATE	可为空
	category	VARCHAR (2)	可为空
	uid	INT (11)	外键

接下来，为每个功能规划一下路由、控制器、模型和视图文件，如表 13-3 所示。

表 13-3　详细数据规划

项　　目	数　　据
控制台	
路由	main/dashboard
必选参数	无
可选参数	无
控制器	app/controllers/main/dashboard.php
模型	app/models/incomes.php 和 app/models/expenses.php
视图	app/views/dashboards.php
用户登录和验证	
路由	main/login/action/msg_id
必选参数	无
可选参数	（1）action ，表示用户是否单击“登录”按钮，其可选值如下： do，表示用户已单击“登录”按钮。 （NULL），表示用户未单击“登录”按钮。 （2）msg_id，表示用户登录结果，其可选值如下： 301，表示用户输入的信息不符合要求。 401，表示数据库连接失败，无法登录

续表

项　　目	数　　据
控制器	app/controllers/main/login.php
模型	app/models/user.php
视图	app/views/login.php
用户注册	
路由	main/register/action/msg_id
必选参数	无
可选参数	（1）action，表示用户是否单击“注册”按钮，其可选值如下： do，表示用户已单击“注册”按钮。 （NULL）　，表示用户未单击“登录”按钮。 （2）msg_id，表示用户注册结果，其可选值如下： 201，表示用户已注册成功。 301，表示用户类和姓名类参数不符合要求。 302，表示密码设置不符合要求。 303，表示未上传相片。 304，表示相片上传失败。 305，表示指定的用户名已经存在。 401，表示用户注册失败
控制器	app/controllers/main/register.php
模型	app/models/main/user.php
视图	app/views/register.php
用户注销	
路由	main/logout
必选参数	无
可选参数	无
控制器	app/controllers/main/logout.php
模型	无
视图	无
逐条添加收入/消费记录	
路由	main/add/record_type/action/msg_id
必选参数	record_type，表示欲添加的记录类型，其可选值如下： income，表示收入记录。 expense，表示消费记录
可选参数	（1）action，表示用户是否已经单击“添加收入记录”或“添加消费记录”按钮，其可选值如下： do，表示用户已单击“添加收入记录”或“添加消费记录”按钮。 （NULL） （2）msg_id，表示添加数据记录的结果，其可选值如下： 201，表示数据记录添加成功。 301，表示数据记录描述不符合要求。 302，表示日期格式不正确。 303，表示所涉金额格式不正确。 401，表示添加数据记录的操作失败
控制器	app/controllers/main/add.php

续表

项　　目	数　　据
模型	app/models/incomes.php 和 app/models/expenses.php
视图	app/views/add.php
批量添加收入/消费记录	
路由	main/b_add/action/msg_id
必选参数	无
可选参数	（1）action，表示用户是否已经单击“添加收入记录”或“添加消费记录”按钮，其可选值如下： do，表示用户已单击“添加收入记录”或“添加消费记录”按钮。 （NULL） （2）msg_id，表示批量添加数据记录的结果，其可选值如下： 201，表示数据记录批量添加成功。 301，表示未指定账务文件。 302，表示上传的账务文件格式错误。 401，表示批量添加数据记录的操作失败
控制器	app/controllers/main/b_add.php
模型	app/models/incomes.php 和 app/models/expenses.php
视图	app/views/b_add.php
查看收入/消费记录列表	
路由	main/list/start_year/start_month/start_day/order_type
必选参数	无
可选参数	（1）start_year/start_month/start_day，表示记录的开始时间。 （2）order_type，表示记录排序方式，其可选值如下： asc，表示按记录入库的顺序从前往后排序，最新记录排在最后。 desc，表示按记录入库的顺序从后往前排序，最新记录排在最前
控制器	app/controllers/main/list.php
模型	app/models/incomes.php 和 app/models/expenses.php
视图	app/views/list.php
修改指定的收入/消费记录	
路由	main/modify/record_type/record_id/action/msg_id
必选参数	（1）record_type，表示欲修改的记录类型，其可选值如下： incomes，表示收入记录。 expenses，表示消费记录。 （2）record_id，表示欲修改的记录编号
可选参数	（1）action，表示用户是否已经单击“添加收入记录”或“添加消费记录”按钮，其可选值如下： do，表示用户已单击“添加收入记录”或“添加消费记录”按钮。 （NULL） （2）msg_id，返回修改结果的编号，其可选值如下： 201，表示指定记录修改成功。 301，表示用户输入的数据不符合要求，提示用户重新输入。 401，表示修改操作失败
控制器	app/controllers/main/modify.php
模型	app/models/incomes.php 和 app/models/expenses.php
视图	无

续表

项　　目	数　　据
删除指定的收入/消费记录	
路由	main/delete/record_type/record_id/action/msg_id
必选参数	（1）record_type，表示欲修改的记录类型，其可选值如下： incomes，表示收入记录。 expenses，表示消费记录。 （2）record_id，表示欲修改的记录编号
可选参数	（1）action，表示用户是否已经单击“添加收入记录”或“添加消费记录”按钮，其可选值如下： do，表示用户已单击“添加收入记录”或“添加消费记录”按钮。 （NULL） （2）msg_id，返回修改结果的编号，其可选值如下： 201，表示指定记录修改成功。 301，表示用户输入的数据不符合要求，提示用户重新输入。 401，表示修改操作失败
控制器	app/controllers/main/delete.php
模型	app/models/incomes.php 和 app/models/expenses.php
视图	无

在向框架目录中添加了规划的各种控制器、模型和视图以及使用 Bootstrap UI 框架所需的各种文件之后，框架目录结构如下：

```
|------chapter13
    |------app
        |------controllers
            |------main
                |------add.php
                |------b add.php
                |------dashboard.php
                |------delete.php
                |------list.php
                |------login.php
                |------logout.php
                |------modify.php
                |------register.php
        |------helps
        |------inc
        |------models
            |------expense.php
            |------income.php
            |------user.php
        |------views
            |------add.php
            |------b add.php
            |------dashboard.php
            |------index.php
            |------list.php
            |------login.php
            |------logout.php
```

```
            |------register.php
    |------css
        |------bootstrap.css          //Bootstrap UI 框架的样式表
        |------bootstrap.min.css      //压缩后的 Bootstrap UI 框架样式表
        |------font-awesome.css       //FontAwesome 图标字体样式表
        |------font-awesome.min.css   //压缩后的 FontAwesome 图标字体样式表
        |------morris.css             //Morris 图表插件样式表
        |------overwrite.css          //用来改写 Bootstrap UI 框架样式的样式表

    |------fonts
        |------fontawesome-webfont.eot
        |------fontawesome-webfont.svg
        |------fontawesome-webfont.ttf        //FontAwesome 图标字体样式表
        |------fontawesome-webfont.woff
        |------glyphicons-halflings-regular.eot
        |------glyphicons-halflings-regular.svg
        |------glyphicons-halflings-regular.ttf //Boostrap UI 框架引用的图标字
                                                 体文件
        |------glyphicons-halflings-regular.woff
    |------image
        |------avatar              //用于存放用户上传的头像
    |------js
        |------bootstrap.js        //Bootstrap UI 框架用 JavaScript 脚本文件
        |------bootstrap.min.js//压缩后的 Bootstrap UI 框架用 JavaScript 脚本文件
        |------jquery-2.0.3.min.js//Bootstrap UI 框架依赖的 JavaScript 库
        |------morris.js           //Morris 图表插件 JavaScript 脚本文件
        |------morris.min.js       //压缩后的 Morris 图表插件 JavaScript 脚本文件
        |------upload              //用于存放用户使用批量导入账务功能时上传的账务文件
    |------.htaccess
    |------index.php
    |------kissmvc.php
    |------kissmvc_core.php
```

到这里，我们就完成了数据规划。在下面的编程过程中，一定要严格遵守规划的数据，以便脚本的规模可控、脚本的实现可管理。

13.3.2　用户登录与验证

按照数据规划，本页面主要用于实现用户登录系统时对用户身份的验证工作。涉及到的控制器、模型和视图如表 13-4 所示。

表 13-4　登录页面使用的控制器、模型和视图

路由	main/login/action/msg_id
必选参数	无
可选参数	（1）action ，表示用户是否单击“登录”按钮，其可选值如下： do，表示用户已单击“登录”按钮。 （NULL），表示用户未单击“登录”按钮。 （2）msg_id，表示用户登录结果，其可选值如下： 301，表示用户输入的信息不符合要求。 401，表示数据库连接失败，无法登录

续表

控制器	app/controllers/main/login.php
模型	app/models/user.php
视图	app/views/login.php

现在，我们需要在 KISSMVC 框架的应用目录下的控制器文件夹 main 子文件夹中创建一个名为 login.php 的文件，该文件主要负责用户身份验证的业务逻辑部分。

【例 13.1】 用户登录页面的控制器（app/controllers/main/login.php）。

```
<?php
    function  login($action='',$msgId='') {
        //创建用户登录页面的视图对象，并引用相应的视图文件
        $view = new View(APP_PATH.'views/login.php');

        //定义一些必要的变量
        $msg=array('alert-class'=>'','subject'=>'','body'=>'');
        $username = 'username';
        $password='';

        //判断页面 URL 中是否包含消息码，如果有，则将消息码转换成消息内容
        if($msgId){
            switch($msgId){
                case '301':
                    $msg['alert-class'] = 'alert-warning';
                    $msg['subject'] = 'Warning';
                    $msg['body'] = 'Both the username and password must be
                    specified.';
                    break;
                case '401':
                    $msg['alert-class'] = 'alert-danger';
                    $msg['subject'] = 'Error';
                    $msg['body'] = 'No match has been found. Check your
                    inputs, please.';
                    break;
            }
        //若 URL 中没有消息码，继续判断 URL 是否包含动作参数，如果有且为"do"，开始处
          理用户提交数据
        } elseif($action == 'do') {
            //获取用户通过表单提交的数据
            $username = $_POST['username'];
            $password = $_POST['password'];

            //若用户提交的数据为空，则通过在 URL 中指定消息码输出反馈信息
            if(!$username || !$password) {
                header('location:'.WEB_FOLDER.'main/login/do/301');
            } else {
                //若用户提交的数据不为空，则创建一个用户对象
                //然后把用户提交的数据做为条件，通过创建的用户对象查找数据库
                //并尝试获取用户 ID 和全名
                $user = new User();
                $user->retrieve_one('username = ? AND password = ?',
                array($username, $password));
                $uid = $user->get('uid');
                $fullname = explode(' ', $user->get('fullname'));

                //若成功获取到用户 ID，表示数据库中存在该用户
                //因此，将用户的 ID 和命名存入相应的 SESSION 变量中，并转向控制台页面
```

```
                if($uid) {
                    $_SESSION['uid'] = $uid;
                    $_SESSION['firstname'] = $fullname[0];
                    header('location:'.WEB_FOLDER.'main/dashboard');
                } else {
                    //若获取不到用户 ID，表示数据库中不存在匹配用户
                    //通过在 URL 中指定消息码输出反馈信息
                    header('location:'.WEB_FOLDER.'main/login/do/401');
                }
            }
        }

        //将用户信息和反馈信息输出到之前创建的视图对象中
        $view->set('username', $username);
        $view->set('alert_class', $msg['alert-class']);
        $view->set('subject', $msg['subject']);
        $view->set('body', $msg['body']);
        $view->dump();
    }
?>
```

看完这段脚本，大家应该会发现，我们把验证用户身份的业务逻辑全部写在了控制器中，并通过控制器调用了 USER 模型和 app/views/login.php 视图文件。通过控制器和模型的交互，获取到了视图文件所需要的所有变量。最后，使用视图对象的 set 方法将获取到的所有变量输出到视图文件中。

接下来，我们在应用目录下的模型文件夹中创建一个名为 user.php 的文件。该文件描述了数据库中的 user 表的结构。脚本如下：

【例 13.2】 用户表模型（app/models/user.php）。

```
<?php
   class User extends Model {
    function User() {
        parent::__construct('uid','user','getdbh');
        $this->rs['uid'] = '';
        $this->rs['username'] = '';
        $this->rs['password'] = '';
        $this->rs['fullname'] = '';
        $this->rs['gender'] = '';
        $this->rs['avatar'] = '';
        $this->rs['created_dt'] = '';
    }
   }
?>
```

在这段脚本中，我们定义了一个名为 User 的类，并在 User 类中定义了一个同名方法；然后通过引用其父类的__construct 方法初始化了 User 类；接着，把 user 数据表中的各字段做为索引添加到了 User 对象的 rs 属性中。

最后，在应用目录下的 views 文件夹中创建一个名为 login.php 的文件，该文件描述了用户登录页面的布局。我们要将需要动态替换的内容定义成变量，这样一来，就可以在控制器中对这些变量进行设置。由于篇幅所限，视图文件的内容就不在这里展示了。大家一定要记住的一点是，一定不要在视图文件中定义任何业务逻辑，也就是说不要在视图文件中做任何条件判断。因为在视图文件中做条件判断会严重降低视图文件的可维护性。图 13-8 所示展示了用户登录页面的布局。

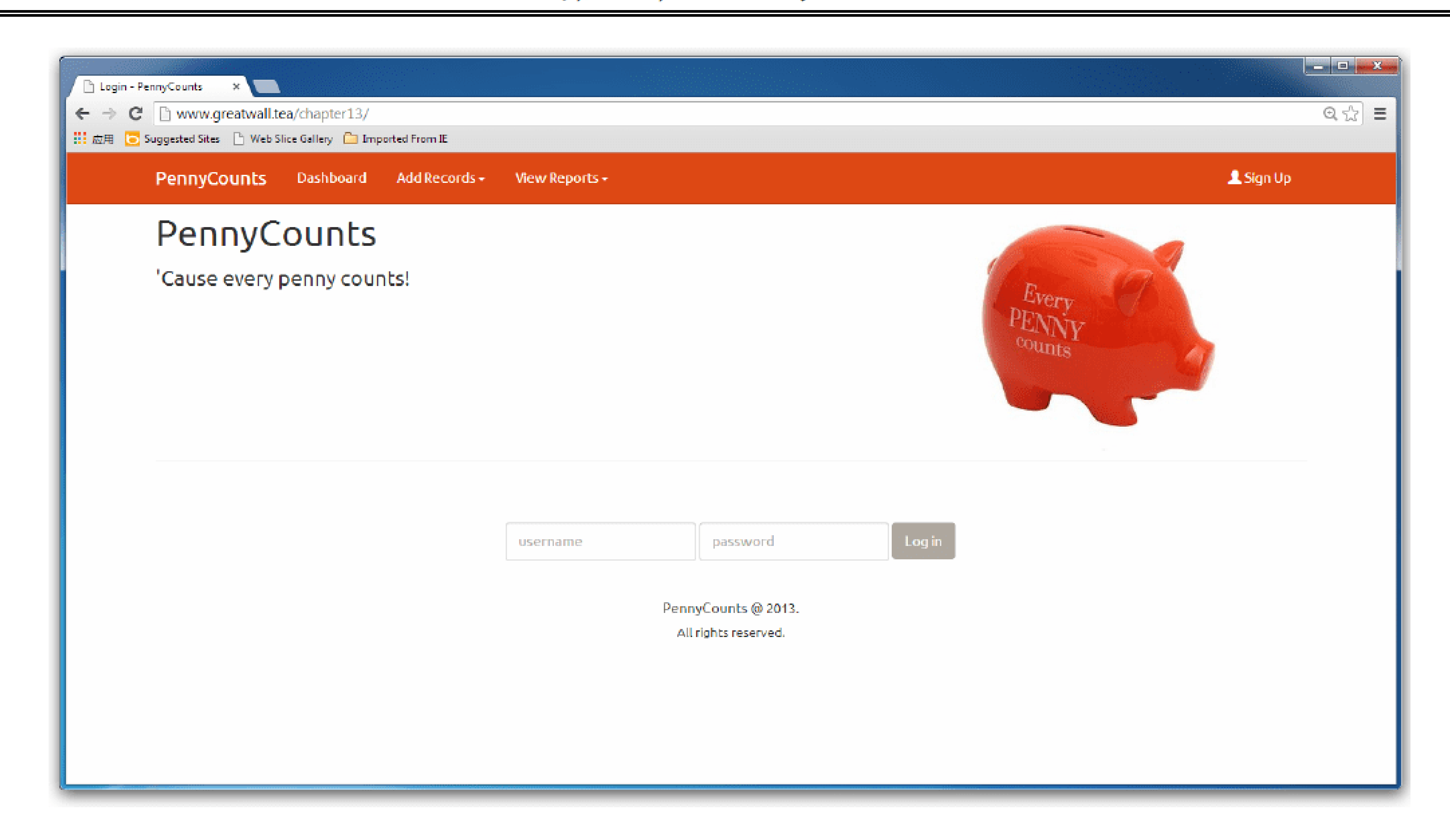


图 13-8　用户登录页面

13.3.3　用户注册

按照数据规划，本页面主要用于实现用户的注册。涉及到的控制器、模型和视图文件如表 13-5 所示。

表 13-5　注册页面使用的控制器、模型和视图

路由	main/register/action/msg_id
必选参数	无
可选参数	（1）action，表示用户是否单击“注册”按钮，其可选值如下： do，表示用户已单击“注册”按钮。 （NULL），表示用户未单击“登录”按钮。 （2）msg_id，表示用户注册结果，其可选值如下： 201，表示用户已注册成功。 301，表示用户类和姓名类参数不符合要求。 302，表示密码设置不符合要求。 303，表示未上传相片。 304，表示相片上传失败。 305，表示指定的用户名已经存在。 401，表示用户注册失败
控制器	app/controllers/main/register.php
模型	app/models/main/user.php
视图	app/views/register.php

现在，我们需要在应用目录下的控制器文件夹下的 main 子文件夹中创建一个名为 register.php 的文件，它定义了用户注册的业务逻辑，其脚本如下：

【例 13.3】 用户注册页面的控制器（app/controllers/main/register.php）。

```
<?php
    function _register($action="", $msgId=""){
        $view = new View(APP_PATH.'views/register.php');

        $msg=array('alert-class'=>'alert-default','subject'=>'','body'=>
        '');

        if($msgId) {
            switch($msgId) {
                case '201':
                    $msg['alert-class'] = 'alert-success';
                    $msg['subject'] = 'Success!';
                    $msg['body'] = "Congratulations! You've been one of us,
                    finally!";
                    break;
                case '301':
                    $msg['alert-class'] = 'alert-warning';
                    $msg['subject'] = 'Warning!';
                    $msg['body'] = 'The username, first name, and last name
                    each
                                must be a string of 3 to 24 characters.';
                    break;
                case '302':
                    $msg['alert-class'] = 'alert-warning';
                    $msg['subject'] = 'Warning!';
                    $msg['body'] = 'The password must be a string of 7 to
                    15 characters,
                                and ensure that you have enterred the same
                                password twice.';
                    break;
                case '303':
                    $msg['alert-class'] = 'alert-warning';
                    $msg['subject'] = 'Warning!';
                    $msg['body'] = 'You have to upload a photo of yourself.';

                    break;
                case '304':
                    $msg['alert-class'] = 'alert-warning';
                    $msg['subject'] = 'Warning!';
                    $msg['body'] = 'The image fails to be uploaded. Please
                    contact the administrator.';
                    break;
                case '305':
                    $msg['alert-class'] = 'alert-warning';
                    $msg['subject'] = 'Warning!';
                    $msg['body'] = 'The username you specified already
                    exists. Please enter another one.';
                    break;

                case '401':
                    $msg['alert-class'] = 'alert-error';
                    $msg['subject'] = 'Error!';
                    $msg['body'] = 'Something goes wrong with our database!
                                Please sign youself up some day in the near
                                future!';
                    break;
            }
        } elseif ($action == 'do') {
            //获取用户提交的数据
```

```
        $username = $_POST['username'];
        $firstname = $_POST['firstname'];
        $lastname = $_POST['lastname'];
        $password = $_POST['password'];
        $confirm = $_POST['confirm'];
        $gender = $_POST['gender'];

        $user = new User();
        $user->retrieve_one('username=?', array($username));
        $uid = $user->get('uid');


        //验证用户输入数据
        $fault_id = 0;
        if(!auth_strings(array($username,$firstname,$lastname),2, 25)) {
            $fault_id = 1;
        } elseif(!auth_strings(array($password, $confirm), 6, 16)) {
            $fault_id = 2;
        } elseif(!auth_image_file($_FILES['photo'])) {
            $fault_id = 3;
        } elseif($uid) {
            $fault_id = 5;
        }

        //若用户输入的信息有误，则转向相应警告页面，否则开始上传文件
        if($fault_id > 0){
        header('location:'.WEB_FOLDER.'main/register/do/30'.$fault_id);
    } else {
        //上传文件
        $photo_dest="\\img\\avatar\\".base64_encode($username).".".
                    substr($_FILES['photo']['type'],6);
        move_uploaded_file($_FILES['photo']['tmp_name'],ABSOLUTE_WEB_
        ROOT.$photo_dest);

        //判断已上传的文件是否存在，若存在，则开始写入数据库，否则转向相应警告页面
        if(file_exists(ABSOLUTE_WEB_ROOT.$photo_dest)){
            //开始写入数据库
            $user->set('username', $username);
            $user->set('fullname', $firstname.' '.$lastname);
            $user->set('gender', $gender);
            $user->set('password', $password);
            $user->set('avatar', $photo_dest);
            $user->set('created_dt',date('Y-m-d'));
            $user->create();
            $uid = $user->get('uid');
            if($uid) {
                $_SESSION['uid'] = $uid;
                $_SESSION['firstname'] = $firstname;
                header('location:'.WEB_FOLDER.'main/register/do/201');
            } else {
                header('location:'.WEB_FOLDER.'main/register/do/401');
                }
            } else {
            header('location:'.WEB_FOLDER.'main/register/do/304');
            }
        }
    }

    $view->set('alert_class', $msg['alert-class']);
```

```
        $view->set('subject', $msg['subject']);
        $view->set('body', $msg['body']);
        $view->dump();
    }
?>
```

在上面这段脚本中，业务逻辑的实现与用户登录页面类似，都是先判断是否存在消息码。若存在，则将消息码转换成相应的消息，然后输出到页面；若不存在，则继续判断用户是否提交了表单。若提交了，则开始获取并验证用户数据。

值得注意的是，我们在处理用户上传的照片文件时，使用了一个名为 ABSOLUTE_WEB_ROOT 的常量，该常量存储的是框架根目录在服务器上的绝对路径。大家如果也想定义一些常量，可以在框架根目录下的 index.php 文件中添加。

另外，我们在验证用户提交的数据时，使用了三个自定义的函数。如果大家需要在某个控制器中使用自定义的函数，可以直接在该控制器页面的尾部（“?>”）前添加。我们在用户注册页面上定义的三个用于验证用户提交的数据的函数如下：

【例 13.4】 用户注册页面的控制器中的自定义函数（app/controllers/main/register.php）。

```
function auth_strings($strings, $minlen, $maxlen) {
    $false_count = 0;
    foreach ($strings as $str) {
        if(strlen($str) >= $maxlen || strlen($str) <= $minlen)
            $false_count++;
    }

    if($false_count){
        return FALSE;
    } else {
        return TRUE;
    }
}

function auth_passwords($password, $confirm, $minlen, $maxlen){
    if(auth_strings(array($password,$confirm), $minlen, $maxlen) &&
    $password == $confirm){
        return TRUE;
    }  else {
        return FALSE;
    }
}

function auth_image_file($file){
    if($file['name'] && $file['tmp_name'] && preg_match('/image/',
    $file['type'])){
        return TRUE;
    } else {
        return FALSE;
    }
}
```

上面这段脚本中定义了函数实现的功能都很简单，只是可以保证在向数据库中写入用户输出的数据不会出现错误。大家在实际的开发中，应该加强对用户输出数据的验证。

本页面与用户登录页面一样，都引用了 User 模型。关于 User 模型的脚本，大家可以参考例 13.2 的内容。图 13-9 展示了用户注册页面的布局。

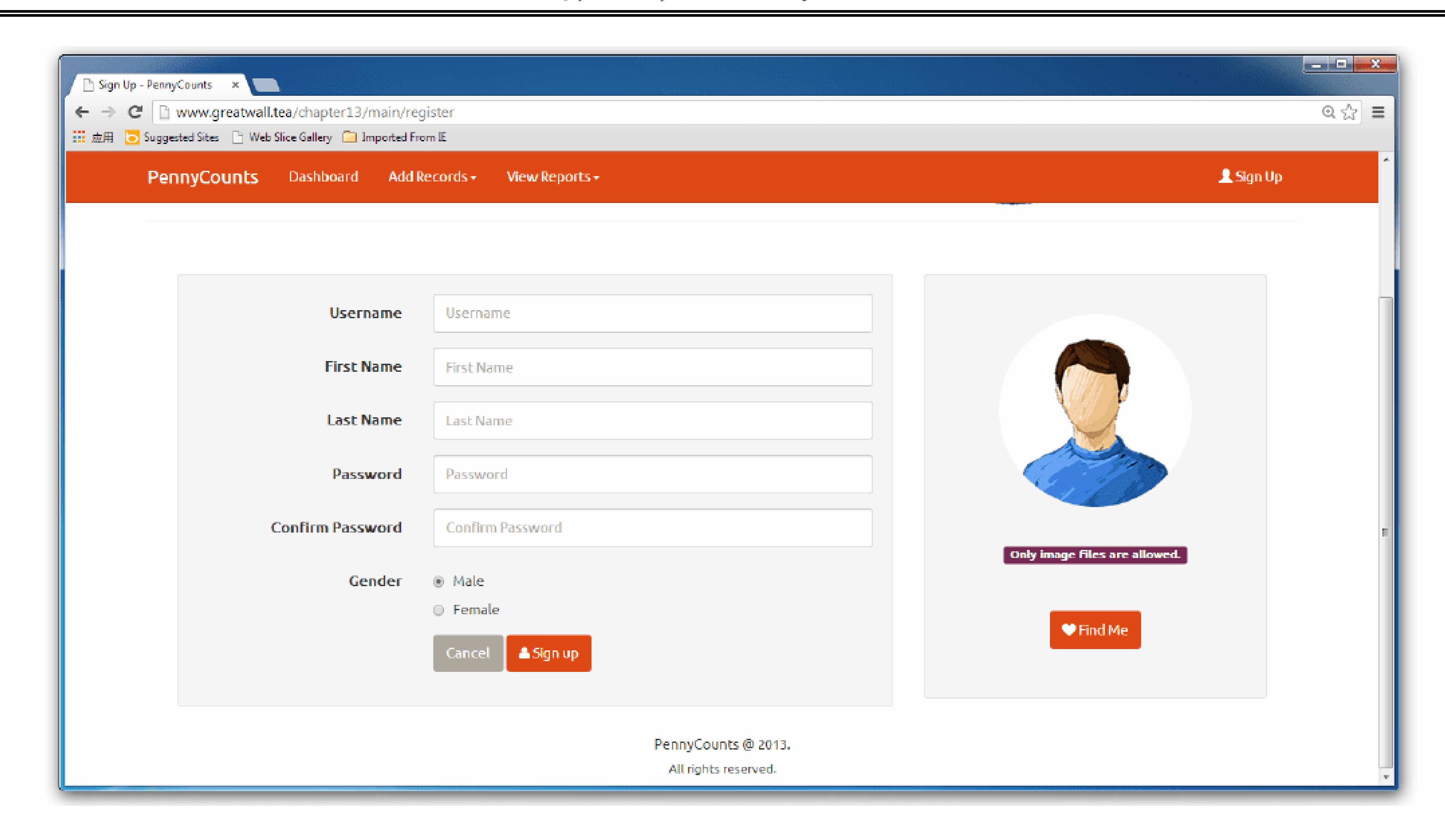


图 13-9　用户注册页面

13.3.4　添加收入和支出记录

按照数据规划，本页面主要用于实现向数据库中添加收入和支出记录。涉及到的控制器、模型和视图文件如表 13-6 所示。

表 13-6　添加收入和支出记录页面使用的控制器、模型和视图

路由	main/add/record_type/action/msg_id
必选参数	record_type，表示欲添加的记录类型，其可选值如下： income，表示收入记录。 expense，表示消费记录
可选参数	（1）action，表示用户是否已经单击“添加收入记录”或“添加消费记录”按钮，其可选值如下： do，表示用户已单击“添加收入记录”或“添加消费记录”按钮。 （NULL），表示该参数未指定，用户未单击上述按钮。 （2）msg_id，表示添加数据记录的结果，其可选值如下： 201，表示数据记录添加成功。 301，表示数据记录描述不符合要求。 302，表示日期格式不正确。 303，表示所涉金额格式不正确。 401，表示添加数据记录的操作失败
控制器	app/controllers/main/add.php
模型	app/models/incomes.php 和 app/models/expenses.php
视图	app/views/add.php

现在，我们需要在应用目录下的控制器文件夹下的 main 子文件夹中创建一个名为 add.php 的文件，它定义了用户向数据库中添加收入和支出记录的业务逻辑，其脚本如下：

【例 13.5】 添加收入和支出记录页面的控制器（app/controllers/main/add.php）。

```
<?php
    function  add($recordType,$action="",$msgId="") {
        //判断用户是否已经登录，未登录则转向登录页面。
        if(!$_SESSION['uid']) {
            header('location:'.WEB_FOLDER);
        } else {
            $view = new View(APP_PATH.'views/add.php');
            $msg = array('alert-class' => '', 'subject' => '', 'body' => '');

            //定义所有可选记录类别
            $category['income'] = array('SW' => 'Salary & Wages',
                                         'BN' => 'Bonus',
                                         'DD' => 'Dividend',
                                         'RT' => 'Rentals',
                                         'OT' => 'Others');
            $category['expense'] = array('GD' => 'Garment & Dressing',
                                          'FD' => 'Food & Vegetable',
                                          'HR' => 'Housing & Rentals',
                                          'UT' => 'Utilities & Others',
                                          'TP' => 'Transportation');

            //定义页面上的标语和相应的可选记录类别
            switch($recordType) {
                case 'income':
                    $slogan = 'Treasuring every penny you earn makes you a
                    rich man...';
                    $category = $category['income'];
                    break;
                case 'expense':
                    $slogan = "Spend with control, being prosperous as you
                    know...";
                    $category = $category['expense'];
                    break;
            }

            //若 URL 中包含$msgId，则根据$msgId 的值来判断展示的内容
            if($msgId)  {
                switch($msgId){
                    case '201':
                        $msg['alert-class'] = 'alert-success';
                        $msg['subject'] = 'Success';
                        $msg['body'] = 'Succeeded in adding the '.
                        $recordType.' record.';
                        break;
                    case '301':
                        $msg['alert-class'] = 'alert-warning';
                        $msg['subject'] = 'Warning';
                        $msg['body'] = 'The description must be a string of
                        5 to 128 characters.';
                        break;
                    case '302':
                        $msg['alert-class'] = 'alert-warning';
                        $msg['subject'] = 'Warning';
                        $msg['body'] = 'The date must be specified in the
                        YYYY-MM-DD format.';
                        break;
                    case '303':
                        $msg['alert-class'] = 'alert-warning';
                        $msg['subject'] = 'Warning';
                        $msg['body'] = 'The amount must be a float number with
                              two bits behind the decimal dot.';
```

```
                    break;
                case '401':
                    $msg['alert-class'] = 'alert-danger';
                    $msg['subject'] = 'Error';
                    $msg['body'] = 'Failed to add the '.$recordType.'
                    record.';
                    break;

                }
            //若 URL 中不包含$msgId，却包含$action，则开始获取并验证用户输入的
              信息
        } elseif($action == 'do'){
            $description = $_POST['description'];
            $date = $_POST['date'];
            $amount = $_POST['amount'];
            $selected_category = $_POST['category'];

            //验证用户输入的信息
            $fault = 0;
            if(strlen($description) > 128 || strlen($description) < 5) {
                $fault = 1;
            } elseif(!auth_date($date)) {
                $fault = 2;
            } elseif(!auth_float($amount)){
                $fault = 3;
            }

            //若验证未通过，展示相应告警信息。若验证通过，则开始将用户输入的信息
              写入数据库
            if($fault){
            header('location:'.WEB_FOLDER.'main/add/'.$recordType. '/do
            /30'.$fault);
            } else {
                switch($recordType){
                    case 'income':
                        $income = new Income();
                        $income->set('description', $description);
                        $income->set('amount', $amount);
                        $income->set('add_dt',$date);
                        $income->set('mod_dt',$date);
                        $income->set('category',$selected_category);
                        $income->set('uid',$_SESSION['uid']);
                        $income->create();
                        $incid = $income->get('incid');
                        if($incid) {
                            header('location:'.WEB_FOLDER.'main/add/
                            income /do/201');
                        } else {
                            header('location:'.WEB_FOLDER.'main/add/
                        income/do/401');
                        }
                        break;

                    case 'expense':
                        $expense = new Expense();
                        $expense->set('description', $description);
                        $expense->set('amount', $amount);
                        $expense->set('add_dt',$date);
                        $expense->set('mod_dt',$date);
```

```
                            $expense->set('category',$selected
                            category);
                            $expense->set('uid',$_SESSION['uid']);
                            $expense->create();
                            $expid = $expense->get('expid');
                            if($expid) {
                                header('location:'.WEB FOLDER.'main/add
                                /expense/do/201');
                            } else {
                                header('location:'.WEB FOLDER.'main/add
                                /expense/do/401');
                            }
                            break;
                    }
                }
            }
        }

        $view->set('recordType', $recordType);
        $view->set('slogan', $slogan);
        $view->set('alert_class', $msg['alert-class']);
        $view->set('subject', $msg['subject']);
        $view->set('body', $msg['body']);
        $view->set('category', $category);
        $view->dump();
    }
?>
```

在上面这段脚本中，我们为了验证用户提交的数据，也定义了两个自定义函数，其脚本如下：

```
function auth_date($date_str){
    $reg exp  =  "/^((((1[6-9]|[2-9]\d)\d{2})-(0?[13578]|1[02])-(0?[1-9]
|[12]\d|3[01]))|
  (((1[6-9]|[2-9]\d)\d{2})-(0?[13456789]|1[012])-(0?[1-9]| [12]\d|30))|
  (((1[6-9]|[2-9]\d)\d{2})-0?2-(0?[1-9]|1\d|2[0-8]))|(((1[6-9]|[2-9]
  \d)(0[48]|[2468][048]|
    [13579][26])|((16|[2468][048]|[3579][26])00))-0?2-29-))$/";
  if(preg match($reg exp, $date str)) {
        return TRUE;
  } else {
        return FALSE;
  }
}

function auth float($float) {
    $reg exp = "/^\d{1,7}(\.\d{1,2})?$/";
    if(preg_match($reg_exp, $float)){
        return TRUE;
    } else {
        return FALSE;
    }
}
```

这两个验证函数都使用了正则表达式。其中用于验证日期格式的正则表达式比较复杂，大家可以保存下来，以便后续使用。图 13-10 和图 13-11 分别展示了添加收入和支出记录页面的布局。

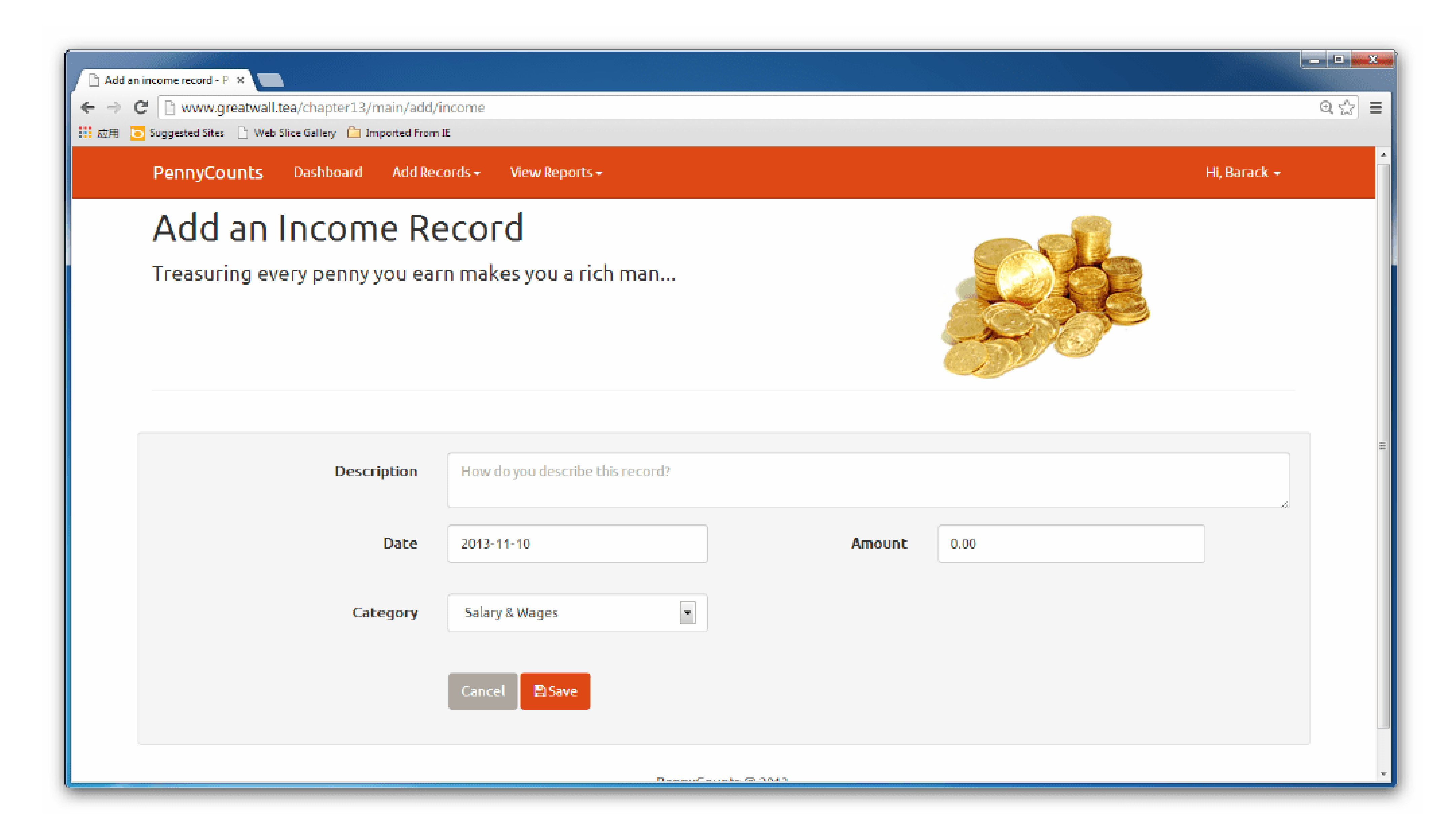


图 13-10　添加收入记录

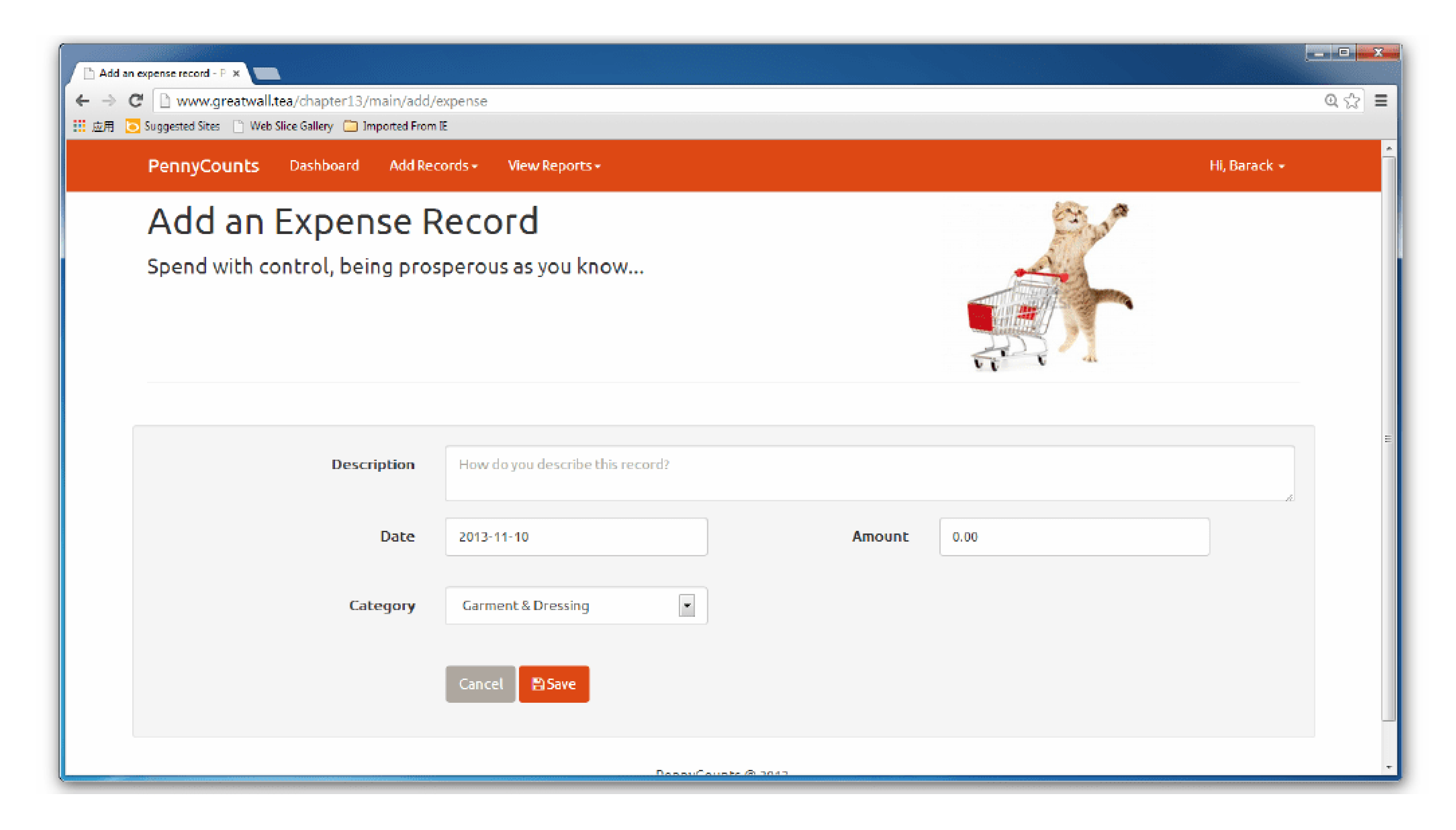


图 13-11　添加支出记录

在向数据库中添加收入和支出记录的过程中，我们还引用了 Income 类和 Expense 类。为了定义这两个类，需要在应用目录下的模型文件夹中创建两个名为 income.php 和 expense.php 的文件，其脚本如例 13.6 和例 13.7 所示。

【例 13.6】 收入记录表模型（app/models/income.php）。

```
<?php
    class Income extends Model {
        function Income() {
            parent::  construct('incid','income','getdbh');
            $this -> rs['incid'] = '';
            $this -> rs['description'] = '';
            $this -> rs['amount'] = '';
```

```
            $this -> rs['add dt'] = '';
            $this -> rs['mod dt'] = '';
            $this -> rs['category'] = '';
            $this -> rs['uid'] = '';
        }
    }
?>
```

【例 13.7】 支出记录表模型（app/models/expense.php）。

```
<?php
    class Expense extends Model {
        function Expense() {
            parent::  construct('expid','expense','getdbh');
            $this -> rs['expid'] = '';
            $this -> rs['description'] = '';
            $this -> rs['amount'] = '';
            $this -> rs['add dt'] = '';
            $this -> rs['mod dt'] = '';
            $this -> rs['category'] = '';
            $this -> rs['uid'] = '';
        }
    }
?>
```

至此，我们这个记账工具中使用到的所有模型就定义完毕了。

13.3.5　批量添加收入和支出记录

按照数据规划，本页面主要用于实现向数据库中批量添加收入和支出记录。涉及到的控制器、模型和视图文件如表 13-7 所示。

表 13-7　添加收入和支出记录页面使用的控制器、模型和视图

路由	main/b_add/action/msg_id
必选参数	无
可选参数	（1）action，表示用户是否已经单击“添加收入记录”或“添加消费记录”按钮，其可选值如下： do，表示用户已单击“添加收入记录”或“添加消费记录”按钮。 （NULL），表示该参数未指定，用户未单击上述按钮。 （2）msg_id，表示批量添加数据记录的结果，其可选值如下： 201，表示数据记录批量添加成功。 301，表示未指定账务文件。 302，表示上传的账务文件格式错误。 401，表示批量添加数据记录的操作失败
控制器	app/controllers/main/b_add.php
模型	app/models/incomes.php 和 app/models/expenses.php
视图	app/models/views/b_add.php

现在，我们需要在应用目录下的控制器文件夹下的 main 子文件夹中创建一个名为 b_add.php 的文件，它定义了用户向数据库中添加收入和支出记录的业务逻辑，其脚本如下：

【例 13.8】 批量添加收入和支出记录页面的控制器（app/controllers/main/b_add.php）。

```
<?php
    function  b add($action="", $msgId=""){
        $view = new View(APP PATH.'views/b add.php');
        $msg = array('alert-class' => '', 'subject' => '', 'body' => '');
        if(isset($ SESSION['uid'])){
            if($msgId){
                switch($msgId) {
                    case '201':
                        $msg['alert-class'] = 'alert-success';
                        $msg['subject'] = 'Success!';
                        $msg['body'] = "Congratulations! All records in the
                                        template file
                                    has been added to the database!";
                        break;
                    case '301':
                        $msg['alert-class'] = 'alert-warning';
                        $msg['subject'] = 'Warning!';
                        $msg['body'] = 'No template file has been selected.
                                    You must select a file before we can
                                    contine.';
                        break;
                    case '302':
                        $msg['alert-class'] = 'alert-warning';
                        $msg['subject'] = 'Warning!';
                        $msg['body'] = 'The template file you have uploaded
                        is not in the correct format.';
                        break;
                    case '401':
                        $msg['alert-class'] = 'alert-danger';
                        $msg['subject'] = 'Error!';
                        $msg['body'] = 'Something goes wrong with our
                        database! Please try again later!';
                        break;
                }
            } elseif ($action == "do"){
                //若用户提交了表单，则获取用户提交的内容
                $file = $ FILES['template'];
                $name = $file['name'];
                $temp = $file['tmp name'];
                $type = $file['type'];

                //检测用户是否选择了一个文本文件
                if($name && $temp && $type && preg_match('/text/', $type)){

                    //若用户选择了一个文本文件，则开始上传
                    $dest = "\\upload\\".base64 encode($ SESSION['firstname
                    ']).'.txt';
                    move uploaded file($temp, ABSOLUTE WEB ROOT.$dest);

                    //检测文件上传是否成功
                    if(file_exists(ABSOLUTE_WEB_ROOT.$dest)){

                        //若成功，则开始读取文件内容，并将收入记录和消费记录分别
                        //以数组的形式存入两个变量中
                        $records = readTemp(ABSOLUTE_WEB_ROOT.$dest);
                        if(!is_array($records))
                            header('location:'.WEB_FOLDER.'main\b_add\do
                            \302');

                        $incomes = $records['incomes'];
```

```
                        $expenses = $records['expenses'];

                        //检查收入记录是否为空
                        if($incomes){
                            //若收入记录不为空，则开始逐条将收入记录写入数据库
                            $incErrCount = 0;
                            foreach ($incomes as $record) {
                                $income = new Income();
                                $income->set("description",$record
                                ['description']);
                                $income->set("amount",$record['amount']);
                                $income->set("add dt",$record['added dt']);
                                $income->set("mod dt",$record['added dt']);
                                $income->set("category",$record['category']);
                                $income->set("uid",$ SESSION['uid']);
                                $income->create();
                                $incid = $income->get('incid');
                                if(!$incid){
                                    $incErrCount++;
                                }
                            }
                        }

                        //检查消费记录是否为空
                        if($expenses){
                            //若消费记录不为空，则开始逐条将消费记录写入数据库
                            $expErrCount = 0;
                            foreach ($expenses as $record)  {
                                $expense = new Expense();
                                $expense->set("description",$record
                                ['description']);
                                $expense->set("amount",$record['amount']);
                                $expense->set("add_dt",$record
                                ['added_dt']);
                                $expense->set("mod dt",$record
                                ['added dt']);
                                $expense->set("category",$record
                                ['category']);
                                $expense->set("uid",$ SESSION['uid']);
                                $expense->create();
                                $expid = $expense->get('expid');
                                if(!$expid){
                                    $expErrCount++;
                                }
                            }
                        }
                        if(!$incErrCount && !$expErrCount){
                            //若在写入数据库的过程中没有出现错误，则表示操作成功，
                              返回 201 页面
                            header('location:'.WEB FOLDER.'main/b add
                            /do/201');
                        } else {
                            header('location:'.WEB FOLDER.'main/b add
                            /do/401');
                        }
                    }
                } else {
                    header('location:'.WEB FOLDER.'main/b add/do/301');
                }
            }
```

```
        } else {
            header('location:'.WEB FOLDER);
        }

        $view->set('alert class',$msg['alert-class']);
        $view->set('subject',$msg['subject']);
        $view->set('body',$msg['body']);
        $view->dump();
    }
?>
```

在上面的脚本中，我们也使用了一个自定义的函数，用来读取用户上传的账务文件。账务文件的模板，大家可以在框架根目录的 upload 文件夹下找到，其内容如下：

```
...
{E|DE}How do you describe the spending?
{E|AM}How must have you paid?
{E|AD}When did you pay for the things you have just described?
{E|CA}How do you categorize the spending?
...
{I|DE}How do you describe the expense?
{I|AM}How must have you earned?
{E|AD}When did you earn the money you have just described?
{I|CA}How do you categorize the money you have just described?
...
{E|DE}
{E|AM}
{E|AD}
{E|CA}
```

大家可以通过回答模板文件中的问题来填写模板。如果模板文件中的条目数据不够，可以通过复制的方法进行扩展。为了读取用户填写好的账务文件，定义了一个名为 readTemp 的函数。相较于第 10 章的实战练习里定义的那个 readTemp()函数，我们做了一些改进，让其能够将读取到的数据按照既定的格式存入一个数组中。这样一来，便可以通过遍历这个数组来完成向数据库中添加数据记录的任务。脚本的运行效率相比之前更高。该函数的脚本如下：

```
function readTemp($file){
    $fh = fopen($file, 'r');
    $records = '';

    $i = 0;
    while(!feof($fh)){
        $lines[$i] = fgets($fh);
        $i++;
    }
    fclose($fh);

    mb internal encoding('UTF-8');
    if(count($lines)%5 == 0){
        $a=0;
        $b=0;
        for ($i=0; $i < count($lines); $i=$i+5) {
            if(mb substr($lines[$i+1],1,1) == 'I'){
                $records['incomes'][$a]['description'] = trim(mb substr
                ($lines[$i+1], 6));
                $records['incomes'][$a]['amount'] = trim(mb_substr($lines
                [$i+2], 6));
```

```
                $records['incomes'][$a]['added_dt'] = trim(mb_substr
                ($lines[$i+3], 6));
                $records['incomes'][$a]['category'] = trim(mb_substr
                ($lines[$i+4], 6));
                $a++;
            } else {
                $records['expenses'][$b]['description'] = trim(mb_substr
                ($lines[$i+1], 6));
                $records['expenses'][$b]['amount'] = trim(mb_substr
                 ($lines[$i+2], 6));
                $records['expenses'][$b]['added_dt'] = trim(mb_substr
                ($lines[$i+3], 6));
                $records['expenses'][$b]['category'] = trim(mb_substr
                ($lines[$i+4], 6));
                $b++;
            }
        }
    }

    return $records;
}
```

图 13-12 展示了批量添加收入和支出记录的页面布局。

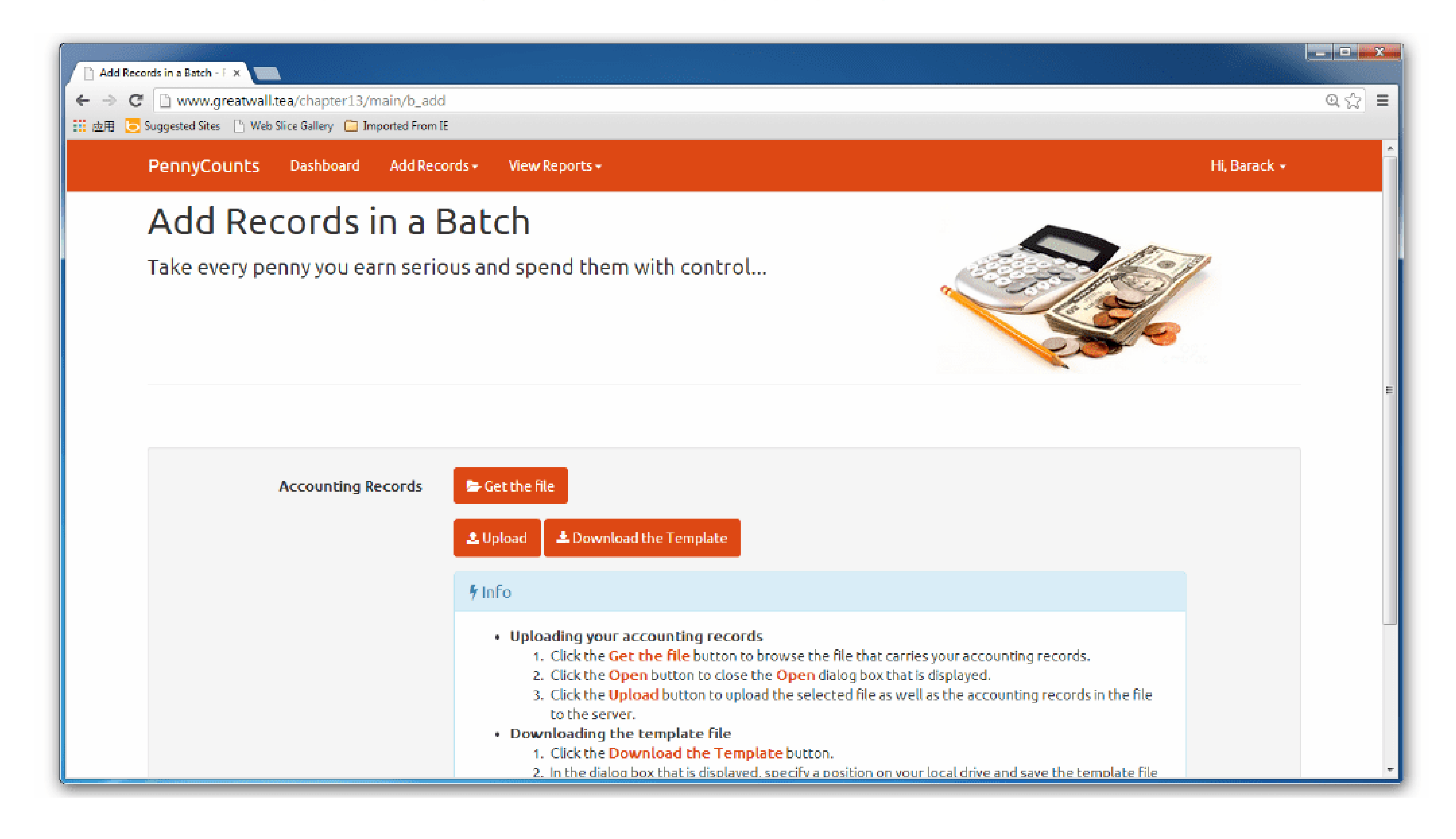


图 13-12　批量添加收入和支出记录

13.3.6　查看数据记录

按照数据规划，本页面主要用于实现从数据库中读取已添加收入和支出记录并以列表的形式展现这些数据。涉及到的控制器、模型和视图文件如表 13-8 所示。

表 13-8　查看数据记录页面使用的控制器、模型和视图

路由	main/list/start_year/start_month/start_day/order_type
必选参数	无

续表

可选参数	（1）start_year/start_month/start_day，表示记录的开始时间。 （2）order_type，表示记录排序方式，其可选值如下： asc，表示按记录入库的顺序从前往后排序，最新记录排在最后。 desc，表示按记录入库的顺序从后往前排序，最新记录排在最前
控制器	app/controllers/main/list.php
模型	app/models/incomes.php 和 app/models/expenses.php
视图	app/views/list.php

现在，我们需要在应用目录下的控制器文件夹下的 main 子文件夹中创建一个名为 list.php 的文件，它定义了从数据库中读取已添加的收入和支出记录的业务逻辑，其脚本如下：

【例 13.9】 查看数据记录页面的控制器（app/controllers/main/list.php）。

```
<?php
    function  list($startYear="",$startMonth="",$startDay="",$sortBy
    ="desc") {
        $view = new View(APP PATH.'views/list.php');
        $records = array(array('id' => '',
                              'type' => '',
                              'description' => '',
                              'income' => '',
                              'expense' => '',
                              'category' => '',
                              'mod date' => ''
        ));

        if(!$ SESSION['uid']){
            header('location:'.WEB_FOLDER);
        } else {
            //获取用户信息
            $user = new User();
            $user->retrieve($ SESSION['uid']);
            $fullname = $user->get('fullname');
            $avatar = $user->get('avatar');

            if($startYear && $startMonth && $startDay){
                $date = $startYear.'-'.$startMonth.'-'.$startDay;
                $startdate = date('M. n, Y',strtotime($date));

                //读取该用户的收入记录
                $income = new Income();
                $rs = $income->retrieve_many('uid = ? AND
                UNIX TIMESTAMP(mod dt) >= UNIX TIMESTAMP(?)', array($ SESSION
                ['uid'],$date));
                $i = 0;
                $totalIncome = 0;
                foreach ($rs as $index => $object) {
                    $records[$i]['id'] = $object->rs['incid'];
                    $records[$i]['type'] = 'income';
                    $records[$i]['description'] =
                    $object->rs['description'];
                    $records[$i]['income'] = $object->rs['amount'];
                    $records[$i]['expense'] = '';
                    $records[$i]['category'] = getCategory($object->rs
                    ['category']);
                    $records[$i]['mod date'] = $object->rs['mod dt'];
                    $i++;
                    $totalIncome = $totalIncome + $object->rs['amount'];
```

```
                }

                //读取该用户的消费记录
                $expense = new Expense();
                $rs = $expense->retrieve_many('uid = ?  AND
                UNIX_TIMESTAMP(mod_dt)>= UNIX_TIMESTAMP(?)', array($_SESSION
                ['uid'],$date));
                $totalExpense = 0;
                foreach ($rs as $index => $object) {
                    $records[$i]['id'] = $object->rs['expid'];
                    $records[$i]['type'] = 'expense';
                    $records[$i]['description'] =
                    $object->rs['description'];
                    $records[$i]['income'] = '';
                    $records[$i]['expense'] = $object->rs['amount'];
                    $records[$i]['category'] = getCategory($object->rs
                    ['category']);
                    $records[$i]['mod date'] = $object->rs['mod dt'];
                    $i++;
                    $totalExpense = $totalExpense + $object->rs['amount'];

                }

                if(($totalIncome - $totalExpense) <= 0) {
                    $balanceClass = 'text-danger';
                } else {
                    $balanceClass = 'text-success';
                }

                //揉合收入和消费记录，并按指定的时间顺序排序
                foreach ($records as $index => $arr) {
                    $dtArr[] = strtotime($arr['mod_date']);
                }

                if ($sortBy == 'desc'){
                    arsort($dtArr);
                } else {
                    asort($dtArr);
                }

                foreach ($dtArr as $idx => $dt) {
                    $sortedRecords[] = $records[$idx];
                }

            }

            $view->set('fullname', $fullname);
            $view->set('avatar', $avatar);
            $view->set('records',$sortedRecords);
            $view->set('totalIncome', $totalIncome);
            $view->set('totalExpense', $totalExpense);
            $view->set('startdate', $startdate);
            $view->set('balanceClass', $balanceClass);
            $view->dump();
        }
    }
?>
```

在上面这段脚本中，大家可以仔细体会一下我们是怎样揉合收入和消费记录，并将它们按指定的时间顺序排序的。弄懂这段脚本可以帮助大家更好地理解数组。

另外，我们在脚本中还使用了一个名为 getCategory()的函数，用来将从数据记录中的 category 字段的值替换成易于理解的短语。其脚本如下：

```
function getCategory($string){
    switch ($string) {
        case 'SW':
            $string = 'Salary & Wages';
            break;
        case 'BN':
            $string = 'Bonus';
            break;
        case 'DD':
            $string = 'Dividend';
            break;
        case 'RT':
            $string = 'Rentals';
            break;
        case 'OT':
            $string = 'Others';
            break;
        case 'GD':
            $string = 'Garment & Dressing';
            break;
        case 'FD':
            $string = 'Food & Vegetable';
            break;
        case 'HR':
            $string = 'Housing & Rentals';
            break;
        case 'UT':
            $string = 'Utilities & Others';
            break;
        case 'TP':
            $string = 'Transportation';
            break;
    }
    return $string;
}
```

图 13-13 展示了查看数据记录页面的布局。

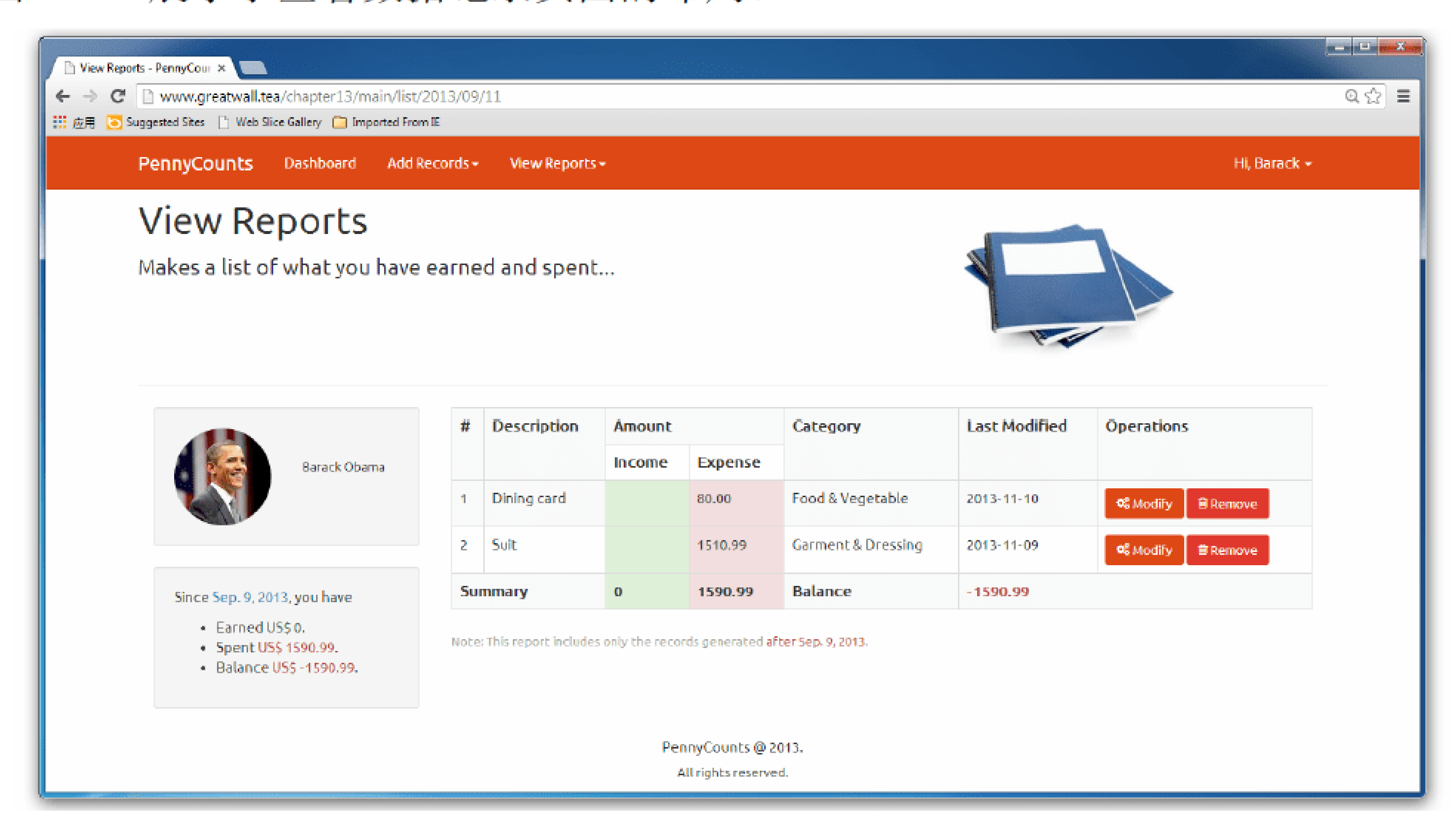


图 13-13　查看数据记录

13.3.7　控制台

按照数据规划，本页面主要用于实现从数据库中读取已添加收入和支出记录并根据这些记录生成相应的统计报表。在这个页面中，我们使用了开源的 jQuery 图表插件——Morris，关于 Morris 插件的使用方法已经超越了本书的范畴，大家可以自行参考其官方网站的内容。本页面涉及到的控制器、模型和视图文件如表 13-9 所示。

表 13-9　控制台页面使用的控制器、模型和视图

路由	main/dashboard
必选参数	无
可选参数	无
控制器	app/controllers/main/dashboard.php
模型	app/models/incomes.php 和 app/models/expenses.php
视图	app/views/dashboards.php

由于这个页面没有任何参数。因此，业务逻辑会比较复杂。我们一起来看一下脚本：

```
<?php
    function _dashboard() {
        $view = new View(APP PATH.'views/dashboard.php');

        if(!isset($ SESSION['uid'])){
            header('location:'.WEB FOLDER);
        } else {
            //获取用户信息
            $user = new User();
            $user->retrieve($ SESSION['uid']);
            $fullname = $user->get('fullname');
            $avatar = $user->get('avatar');


            //计算柱状图区间
            for ($i=0; $i < 12; $i++) {
                if(date('n')-$i == 0) {
                    $month = 12;
                    $year = date('Y') - 1;
                } else {
                    $month = date('n')-$i;
                    $year = date('Y');
                }

                if($month > 9) {
                    $index = $year.'/'.$month;
                    $barData[$index] = array('income'=>0, 'expense'=>0);
                } else {
                    $index = $year.'/0'.$month;
                    $barData[$index] = array('income'=>0, 'expense'=>0);
                }
            }


            //计算总收入、分月收入和各收入分类小计
```

```
$income = new Income();
$rs = $income->retrieve_many('uid = ? AND
                    UNIX_TIMESTAMP(mod_dt)>UNIX_TIMESTAMP(?)',
                    array($_SESSION['uid'],date('Y-m-01',
                    strtotime ('1 year ago'))));
$totalIncome = 0;
$incomeDetails = array('Salary & Wages' => 0,
                       'Bonus' => 0,
                       'Dividend' => 0,
                       'Rentals' => 0,
                       'Others' => 0);
if($rs){
    foreach ($rs as $index => $object) {
        $amount = $object->rs['amount'];
        //总收入
        $totalIncome += $amount;

        //分月收入
        $barData[date('Y/m',strtotime($object->rs['mod_dt']))]
        ['income'] += $amount;

        //各收入分类小计
        switch ($object->rs['category']) {
            case 'SW':
                $incomeDetails['Salary & Wages'] += $amount;
                break;
            case 'BN':
                $incomeDetails['Bonus'] += $amount;
                break;
            case 'DD':
                $incomeDetails['Dividend'] += $amount;
                break;
            case 'RT':
                $incomeDetails['Rentals'] += $amount;
                break;
            case 'OT':
                $incomeDetails['Others'] += $amount;
                break;
        }
    }
}

//计算总支出、分月支出和各收入分类小计
$expense = new Expense();
$rs = $expense->retrieve_many('uid = ? AND
                 UNIX_TIMESTAMP(mod_dt) > UNIX_TIMESTAMP(?)',
                 array($_SESSION['uid'],date('Y-m-01', strtotime
                 ('1 year ago'))));
$totalExpense = 0;
$expenseDetails = array('Garment & Dressing' => 0,
                       'Food & Vegetable' => 0,
                       'Housing & Rentals' => 0,
                       'Utilities & Others' => 0,
                       'Transportation' => 0);
if($rs){
    foreach ($rs as $index => $object) {
        $amount = $object->rs['amount'];
        //总支出
```

```
                    $totalExpense += $amount;

                    //分月支出
                    $barData[date('Y/m',strtotime($object->rs['mod_dt']))]
                    ['expense'] += $amount;

                    //各收入分类小计
                    switch ($object->rs['category']) {
                        case 'GD':
                            $expenseDetails['Garment & Dressing'] += $amount;
                            break;
                        case 'FD':
                            $expenseDetails['Food & Vegetable'] += $amount;
                            break;
                        case 'HR':
                            $expenseDetails['Housing & Rentals'] += $amount;
                            break;
                        case 'UT':
                            $expenseDetails['Utilities & Others'] += $amount;
                            break;
                        case 'TP':
                            $expenseDetails['Transportation'] += $amount;
                            break;
                    }
                }
            }

            $view->set('totalIncome', $totalIncome);
            $view->set('totalExpense', $totalExpense);
            $view->set('balance', $totalIncome-$totalExpense);
            $view->set('barData', $barData);
            $view->set('incomeDetails', $incomeDetails);
            $view->set('expenseDetails', $expenseDetails);
            $view->set('fullname', $fullname);
            $view->set('avatar', $avatar);
            $view->dump();

        }
    }

    function getWhiteSpace($int){
        $string ='';
        for ($i=0; $i < $int; $i++) {
            $string .= ' ';
        }

        return $string;
    }
?>
```

这个页面的主要业务逻辑是，根据 SESSION 变量中缓存的用户 ID，获取用户一年内的所有收入和支出记录，并根据这些记录计算其总收入、总支出、节余及绘制分月收入/支出柱状图和分类小计圈图。大家可以重点关注一下，我们是如何定义柱状图的时间范围、如何计算分月收入以及如何计算分类小计的部分。读懂这几个部分可以帮助我们加深对数据的理解。运行后的效果如图 13-14 所示。

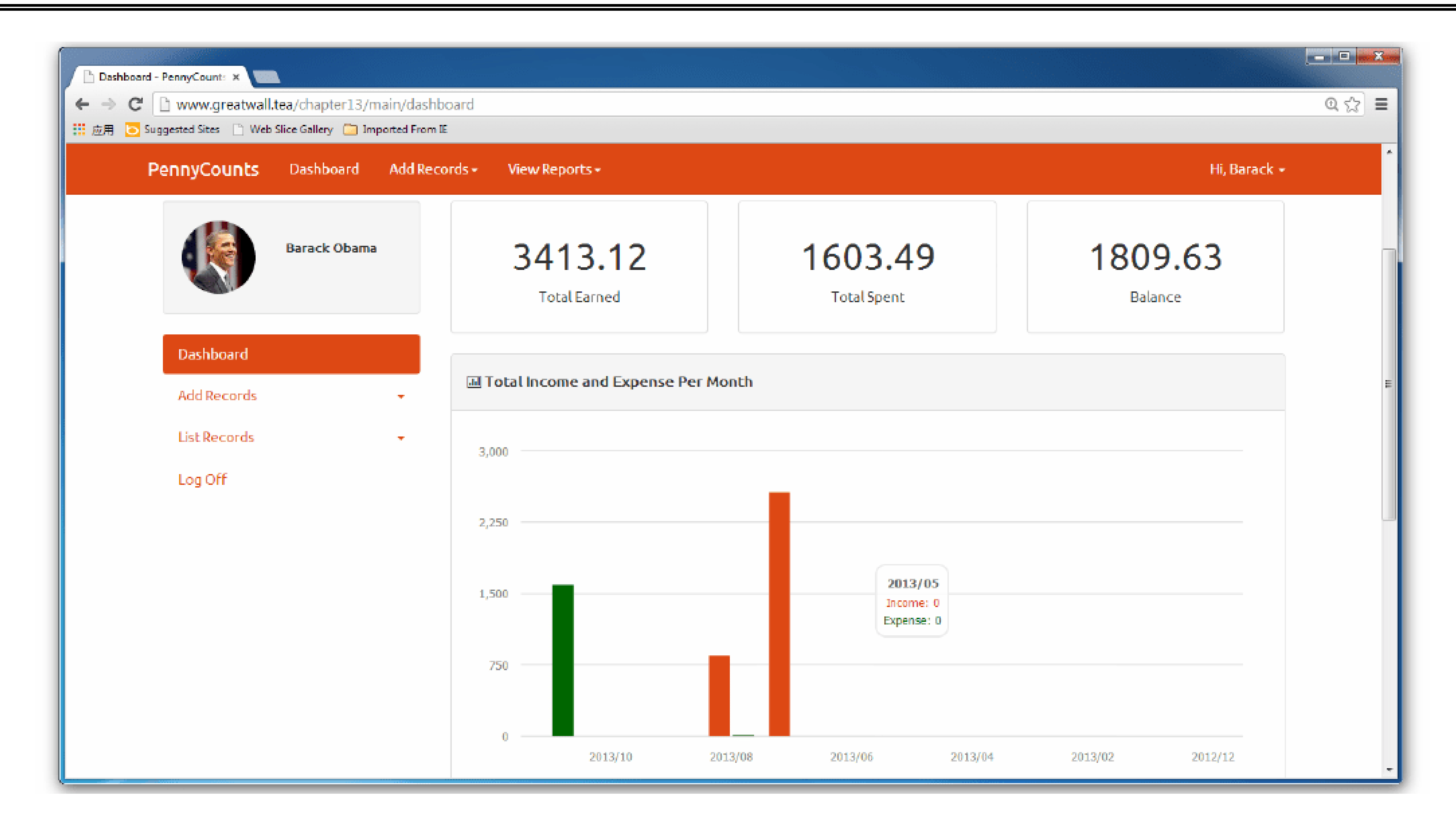


图 13-14　控制台页面的布局

13.4　习　　题

（1）请补充 13.3.1 小节中规划的“修改指定的收入/消费记录”。

（2）请补充 13.3.1 小节中规划的“删除指定的收入/消费记录”。

第 5 篇　开源 PHP 应用

第 14 章　常见开源的 PHP 应用

正是由于 PHP 的开源特性和活跃的社群支持，基于 PHP 的开源应用层出不穷。在本章里，我们就来了解一下当前比较稳定、定期更新的 PHP 应用，这些应用涉及到网络的各方各面。如果有兴趣的话，大家可以在网上搜索一篇名为“Top Ten Open Source PHP Apps”的文章。这篇文章[1]里列出了 PHP 在博客、论坛、内容管理、百科、社交类应用（Digg 和微博客）、图片管理、RSS 和电子商务等方面的应用。在本章里，对博客系统、内容管理系统（CMS）和论坛系统的开源 PHP 应用做相关的介绍。

14.1　WordPress

WordPress 是当今最受欢迎的基于 PHP 的开源博客系统。按照 WordPress 官方网站的介绍，WordPress 也是一种内容管理系统。而维基百科对内容管理系统的定义是“一种用于组织和促进共同内容创造的系统”。对于 WordPress 这个名字的由来，最显而易见的原因就是 Word+Press，Word 代表着构成内容的基本元素，而 Press 则代表着内容创造的方式。通过 Word 和 Press 的结合，实现协同的内容创造。

WordPress 应用程序有两个部分构成，一个为前台，另一个为后台。所谓前台就是广大互联网用户访问利用 WordPress 应用程序搭建的个人博客系统时可以看到的内容，而所谓后台则是只有内容贡献者才可以查看到的部分。前台可以实现的主要功能包括浏览和基于留言的用户交互。前台的表现形式多样，我们可以为前台部署极富个性化的主题，从而使得每一个利用 WordPress 应用程序搭建的个人博客不会千人一面。后台可以实现的主要功能则包括分类管理、博客管理和主题管理等，有着强大的管理功能，可以帮助博客运营者充分展示其想要与大家分享的内容。图 14-1 展示了 WordPress 的后台。

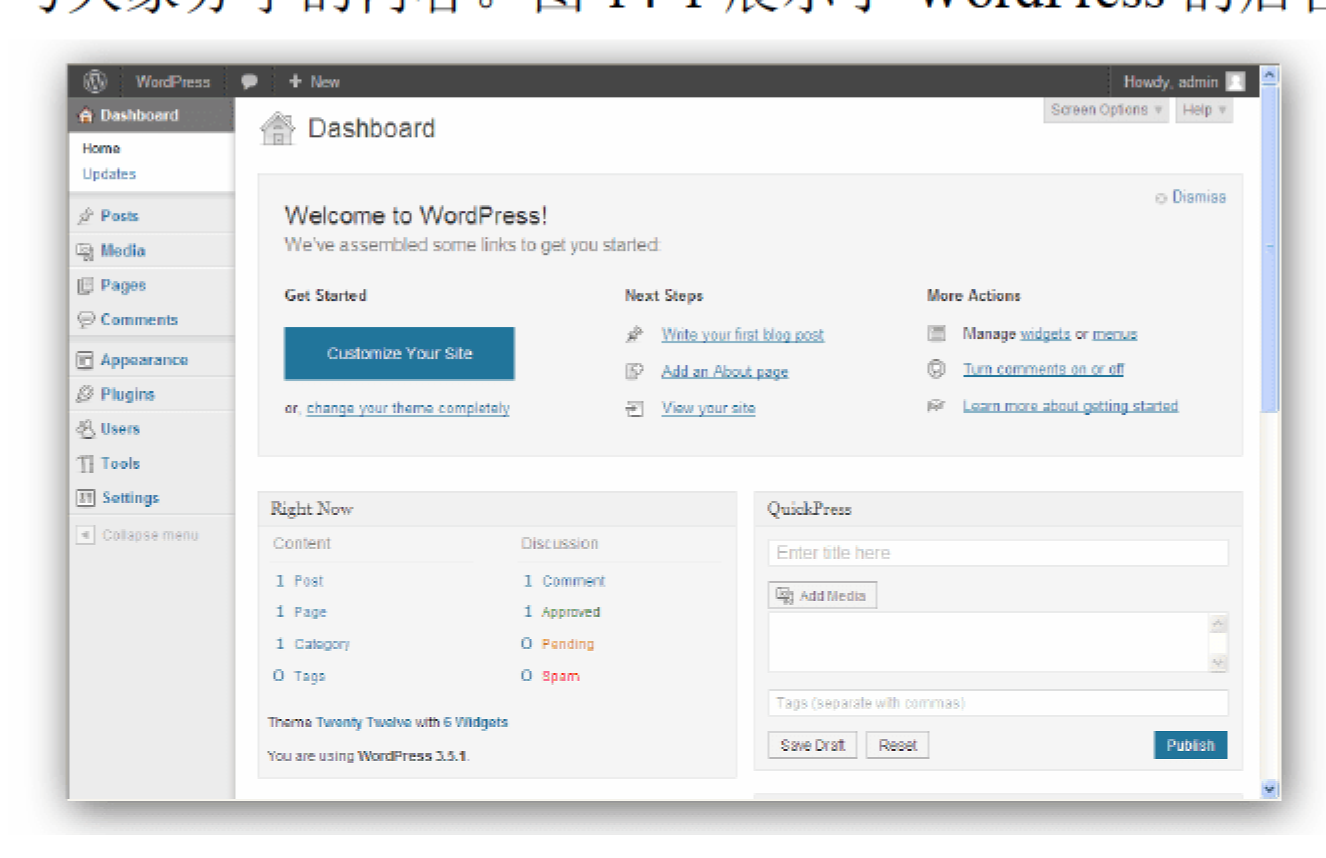


图 14-1　WordPress 的后台

[1] 这篇文章的中文版可以在 http://www.williamlong.info/archives/1852.html 看到。

14.1.1　安装 WordPress

在 WordPress 的官方网站上，我们可以看到关于 WordPress 安装的介绍。如果你和我一样是在本地测试 WordPress 应用程序，那么可以将下载到的 WordPress 包文件（.zip）解压到本地 Web 服务器根文件夹下。

随后打开浏览器，在地址栏中输入 http://localhost。如果在浏览器中看到了类似于如图 14-2 所示的内容，表明 WordPress 可以在本地服务器上安装了。

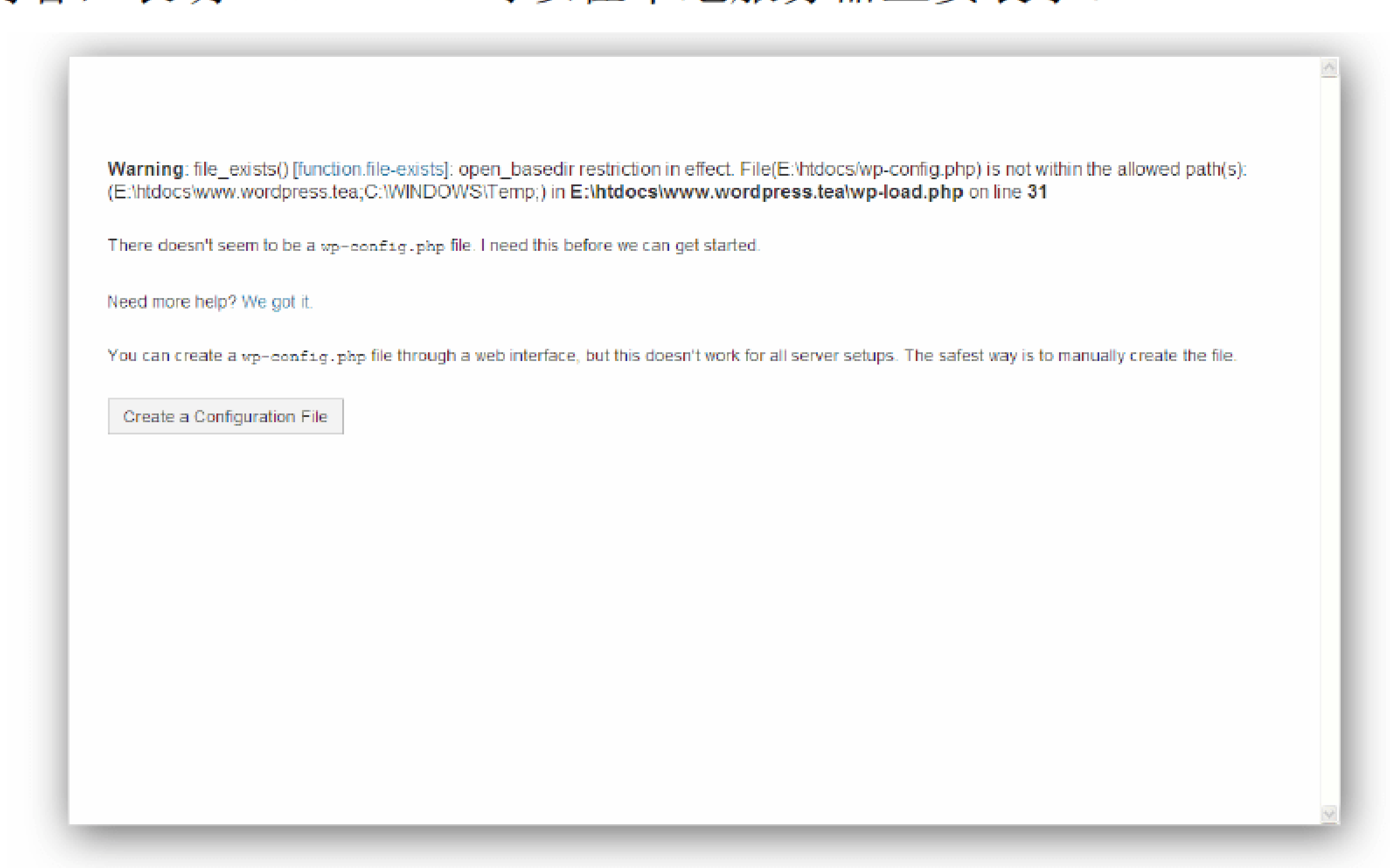


图 14-2　WordPress 安装程序配置文件缺失提示

这时，我们需要先为 WordPress 应用程序配置一个 MySQL 数据库。打开在第 2 章安装的 PHPMyAdmin。使用 root 账户登录后，在首页的“新建数据库”下的文本框内输入“wordpress”，并在其后的下拉列表框内选择“utf8-general-ci”，然后单击“创建”按钮创建数据库。

当界面上提示“创建数据库 Wordpress 成功”时，就可以关闭 PHPMyAdmin 了。

这时，我们回到图 14-2 所示的页面，单击左下角的“Creat a Configuration File”按钮，开始创建 WordPress 配置文件，如图 14-3 所示。

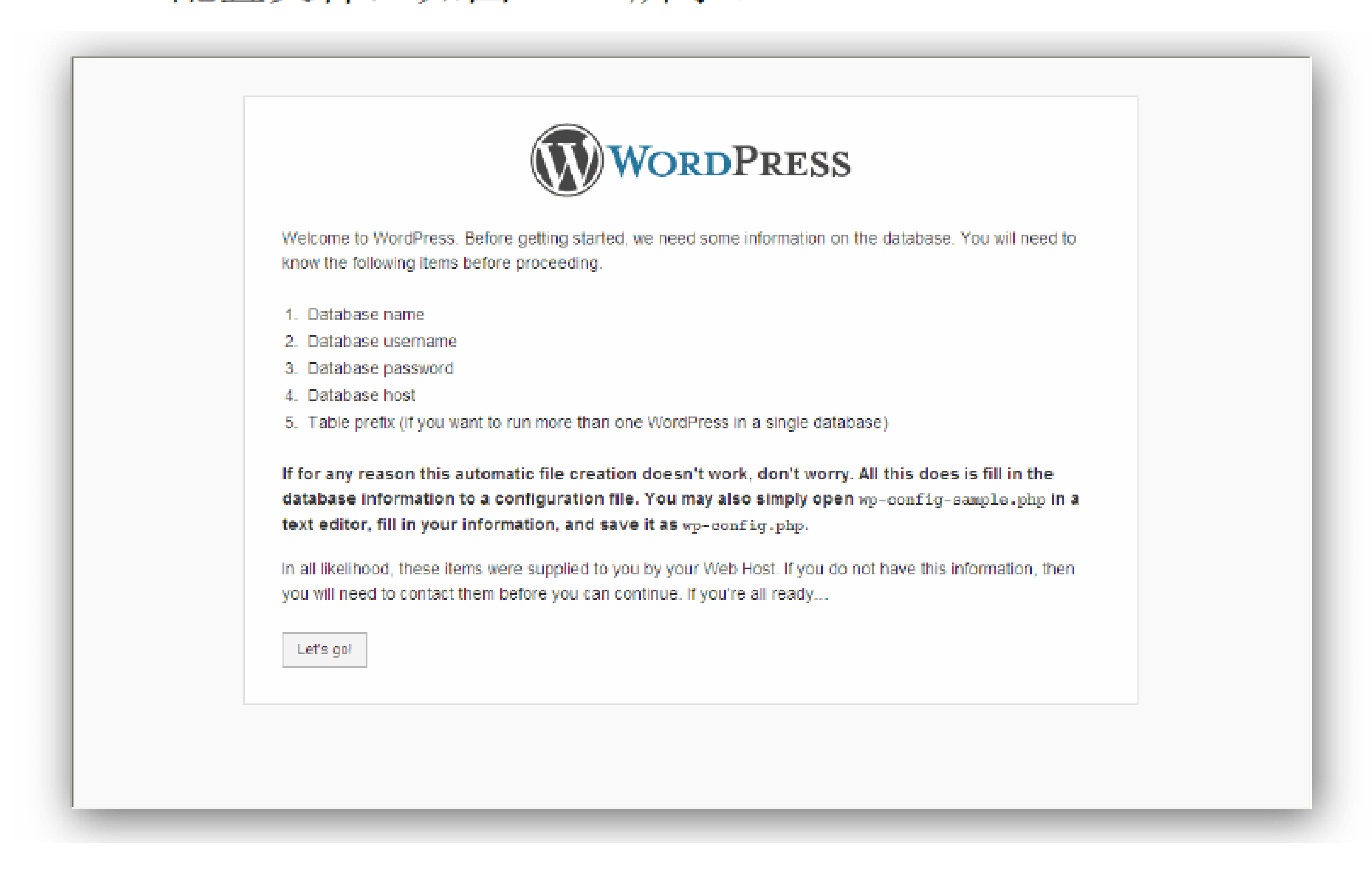


图 14-3　WordPress 安装程序首页

按照首页上的提示，我们需要向服务器提交一些信息。所需信息包括“数据库名”、“数据库用户名”、“数据库密码”、“数据库所在主机”以及“数据库表前缀”。由于我们是在本地进行测试，数据库信息如下。

- ❑ 数据库名：wordpress；
- ❑ 数据库用户名：root；
- ❑ 数据库密码：Passw0rd*；
- ❑ 数据库所在主机：localhost；
- ❑ 数所库表前缀：wp_。

当我们看到如图 14-4 所示的信息时，表明数据库连接成功，可以正式开始安装了。



图 14-4　数据库连接成功

单击“Run the install”按钮，开始安装。当出现如图 14-5 所示的表单时，表明数据库安装完毕，可以开始配置博客信息了。

图 14-5　配置博客信息

在这里，我们需要设置的参数有：“网点名称”、“用户名”、“密码（输入两次）”和“电邮地址”等。由于我们是在本地进行测试，外网无法访问。所以这些信息可以随性输入。

输入的参数分别如下。

- ❑ 网站名称：WordPress；

- 用户名：admin（默认值）；
- 密码：Passw0rd*；
- 电邮：postmaster@localhost.com。

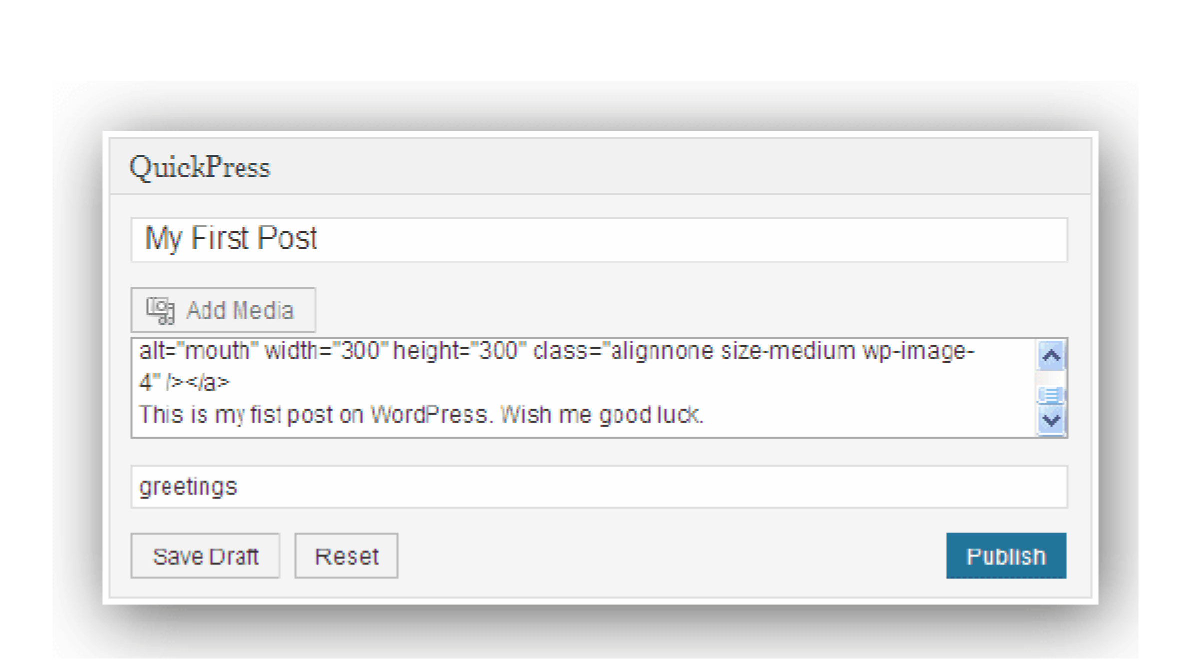
然后单击下方的“Complete”按钮完成配置。随后我们会看到提示成功的页面。单击页面左下角的“Log In”按钮，然后输入之前配置的用户名和密码，即可登录 WordPress 后台。在后台的左上角 WordPress 图标旁边显示了博客的名称“WordPress”。将鼠标指针移上去，然后在下拉菜单中单击“Visit Site”按钮，即可看以站点的前台。

至此，WordPress 的安装成功。

14.1.2　使用 QuickPress 发布一条博客

在首页右下方单击“Site Admin”按钮回到后台，在首页的 Dashboard 上找到“QuickPress”面板。在该面板中有三个文本框，分别为“博客标题”、“博客内容”和“博客标签”。在“博客标题”和“博客内容”两个文本框之间有一个“Add Media”的按钮，单击它可以上传多媒体文件到库中，或直接从库中选择需要插入博客中的文件。在“博客标签”文本框下方有三个按钮 “Save Draft”、“Reset”和“Publish”，分别代表保存草稿、重置和直接发布。

在第一个文本框中输入“My First Post”，然后单击“Add Media”按钮上传了一张图片，接着在下面的文件框里写上一句话，然后再为这条博客打上一个标签，单击“Publish”按钮直接发布博客，如图 14-6 所示。当 QuickPress 面板里出现一条黄底的提示信息说明博客发布成功时，我们可以单击其后的“View Post”链接查看该博客，或者单击“Edit Post”链接修改该博客。



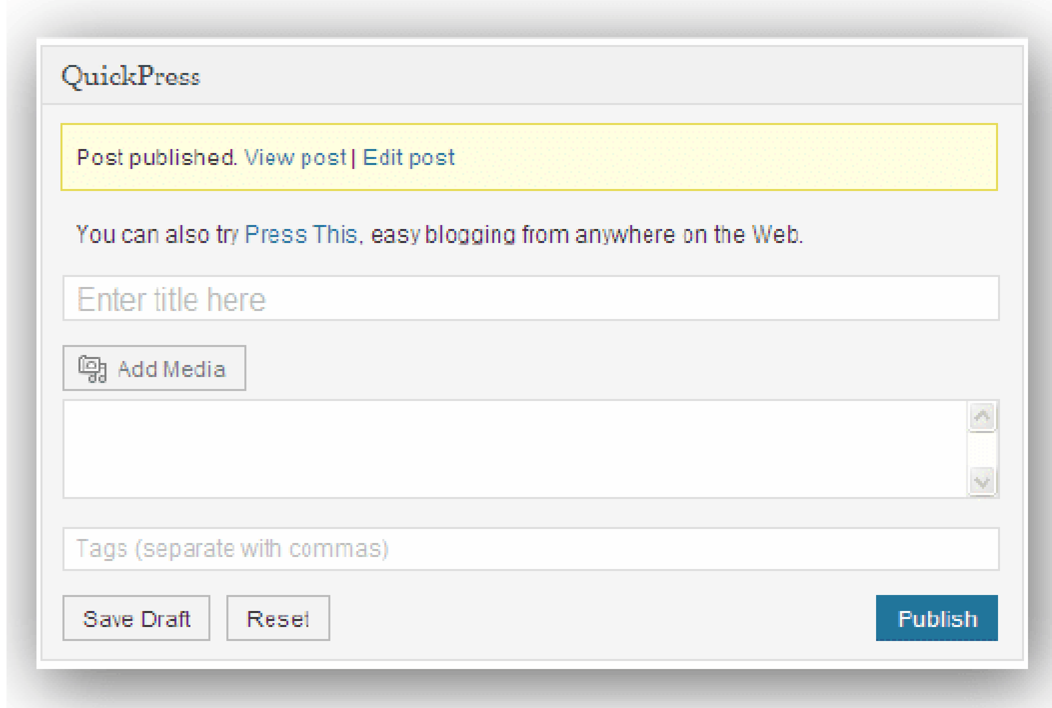


图 14-6　使用 QuickPress 发布一条博客

现在我们单击“View Post”按钮查看已发布的博客，就能在前台看到这条博客的样子了。发布的这条博客如图 14-7 所示。

14.1.3　修改已发布的博客

在查看完刚刚发布的博客之后，发现有些地方不太满意？那我们就回去再改改。单击

页面右侧的“Site Admin”链接回到后台。在后台的左侧有一列菜单，其中第二个名为“Posts”，单击这个菜单按钮，进入“Posts”管理页面，如图 14-8 所示。

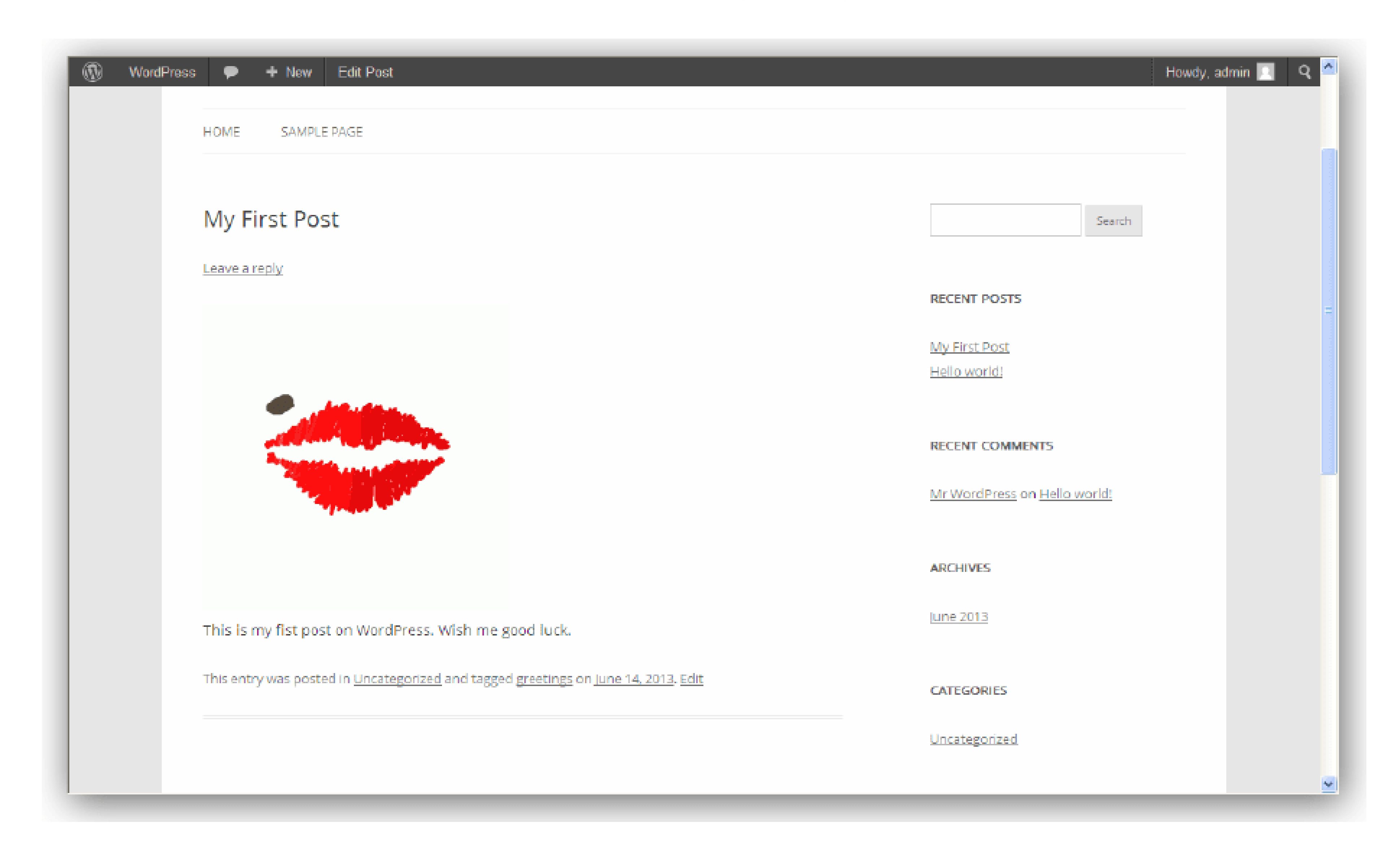


图 14-7　我的第一条博客

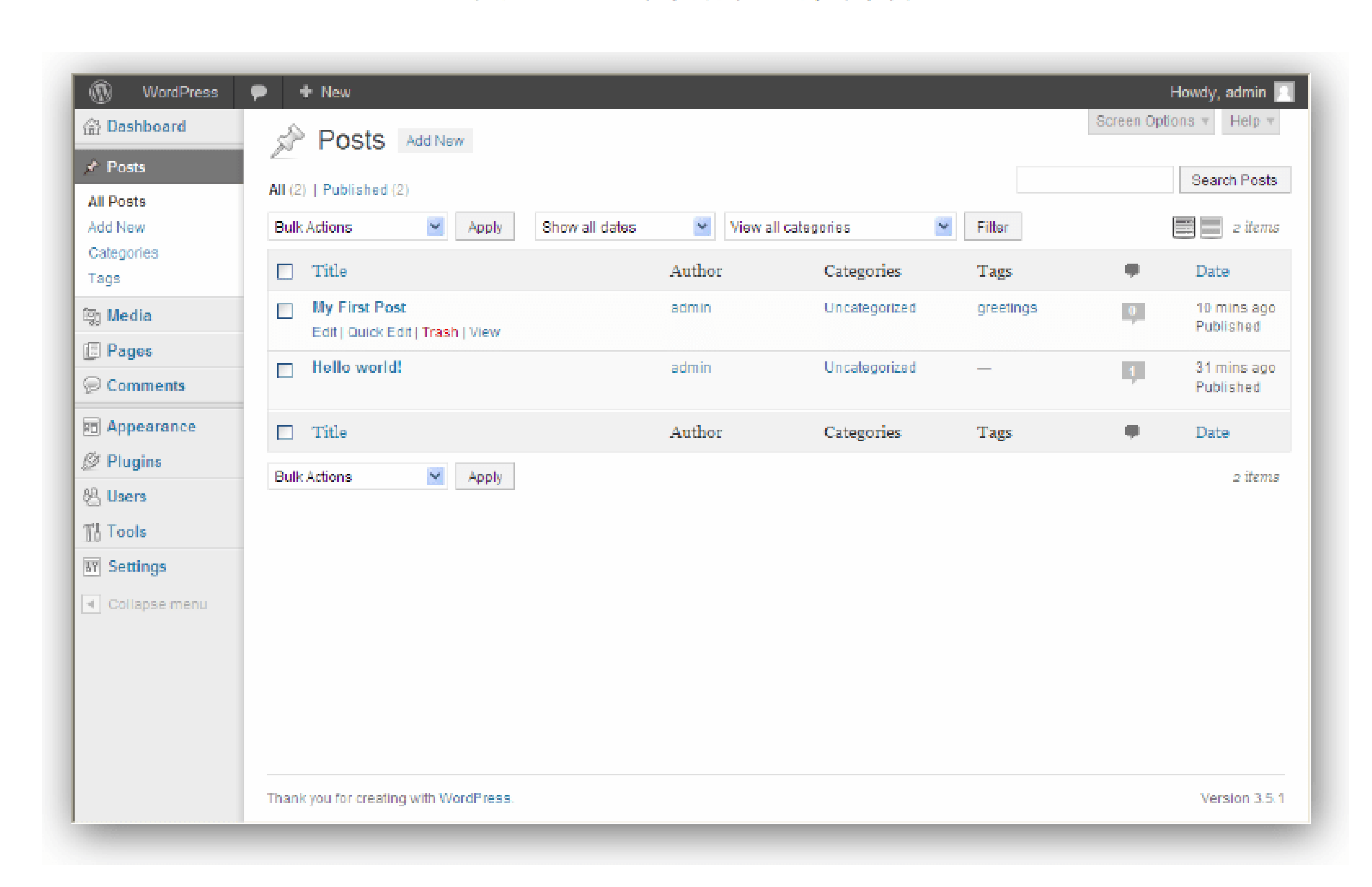


图 14-8　博客管理页面

在这里，我们可以看到一张包含了所有已发布、未发布的博客的列表。现在这张表里有两条博客，第一条就是我们刚刚发布的。鼠标移上去的时候，在这条博客的标题下面出现了四个链接 “Edit”、“Quick Edit”、“Trash”和“View”，分别用于编辑、快速编辑、删除和查看该博客。

在这里我们单击“Edit”编辑该博客，很快便进入了如图 14-9 所示的博客编辑页面。

在该页面上，我们可以修改此博客的任何内容，包括它的名称、内容、所属类别、发布时间、适用范围和格式等等。在这里，我们只修改一下博客的格式：在右侧的“Format”面板中选择“Image”，然后单击上方的“Update”按钮，更新博客。

再次查看该博客时，发现博客上方的大字标题没有了，变成了图片下方的几行小号的

文字，图片在页面中变得更加突出了。

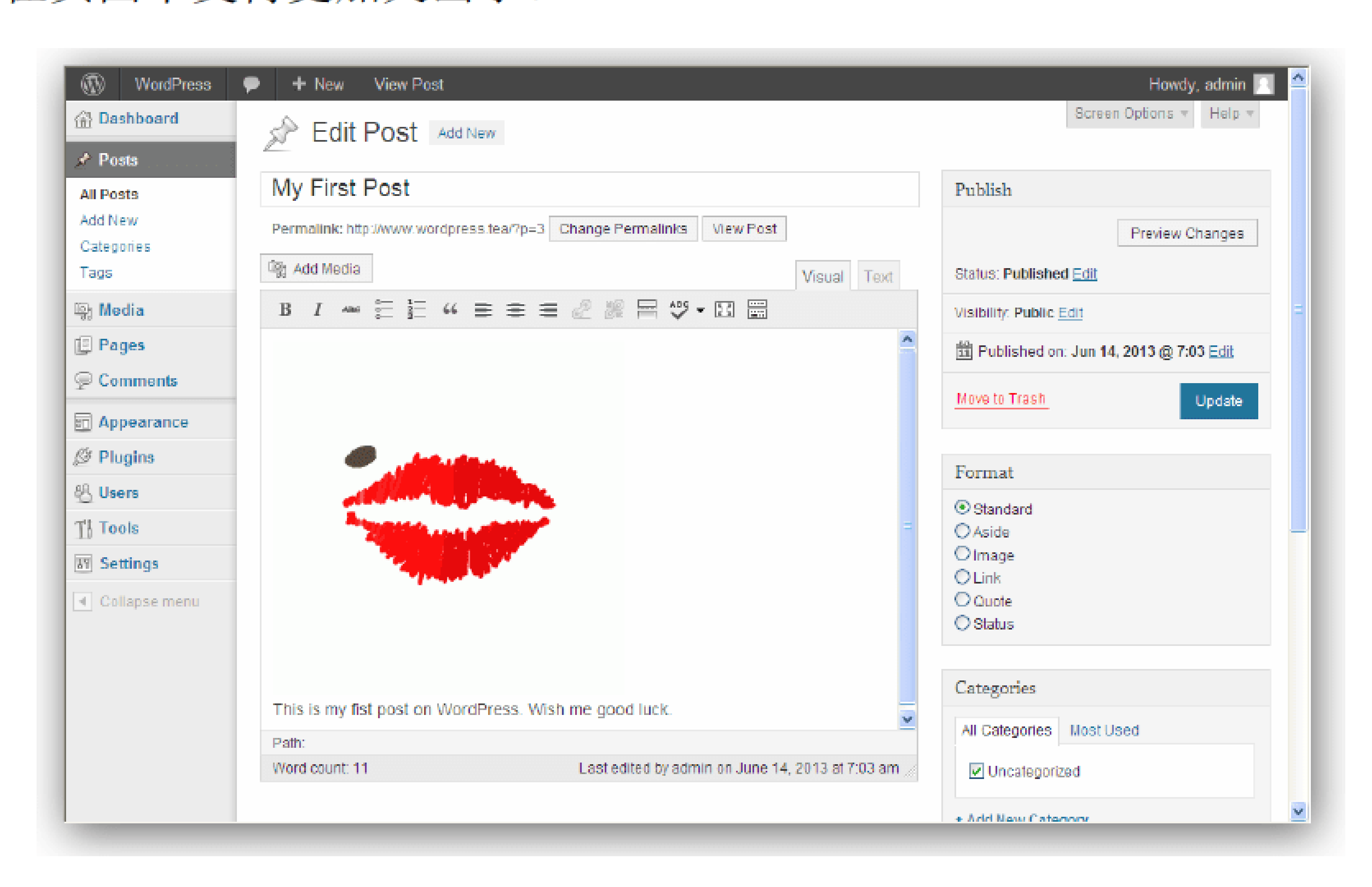


图 14-9　编辑博客

14.1.4　定制页面

在 WordPress 中，与博客类似的元素叫做页面（Page）。在前台看来，页面和博客没有什么不同。但是在后台，页面是可以被直接插入到导航菜单中的，而博客则只能通过其 URL 来实现。为了对定制的页面有个明确的概念，WordPress 安装之后会预置一个名为“Sample Page”的定制的页面，并将其放在了导航菜单中。

单击前台首页导航菜单中的“Sample Page”，会发现它跟我们之前发布的博客的确没有什么区别。回到后台，在页面左侧的导航树中单击“Pages”菜单按钮之后进入页面管理，如图 14-10 所示。在这里，我们可以查看系统中现有的页面，并可以像修改博客一样修改它们。

这里，我们可以把“Sample Page”修改一下，将其名称修改为“Contact Us”。

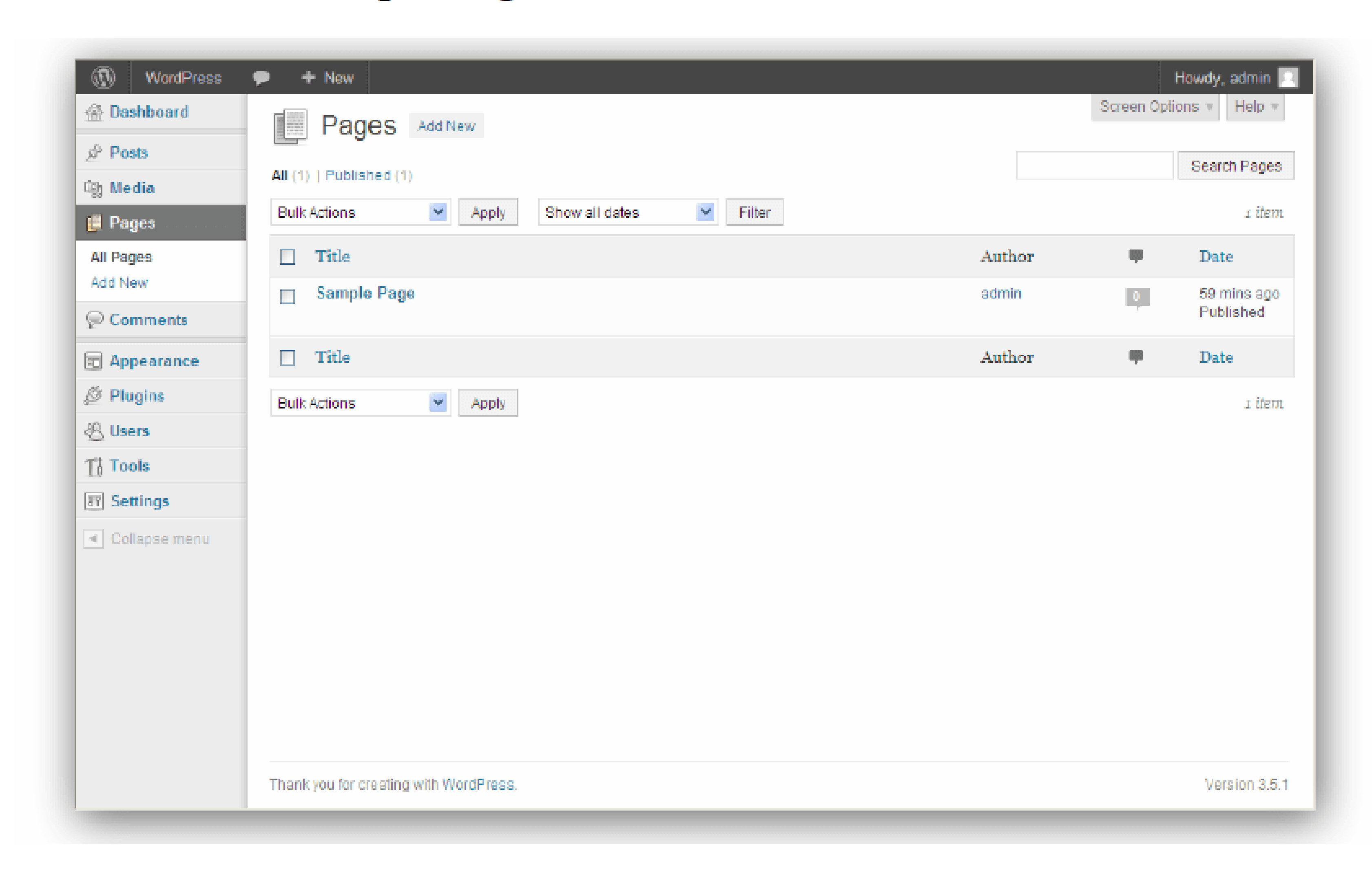


图 14-10　管理页面

将鼠标移动到“Sample Pages”上，随后在出现的四个链接中，单击“Quick Edit”按钮，出现如图 14-11 所示的“快速编辑”表单。这张表单大体上与博客的“快速编辑”相同，如图 14-11 所示。

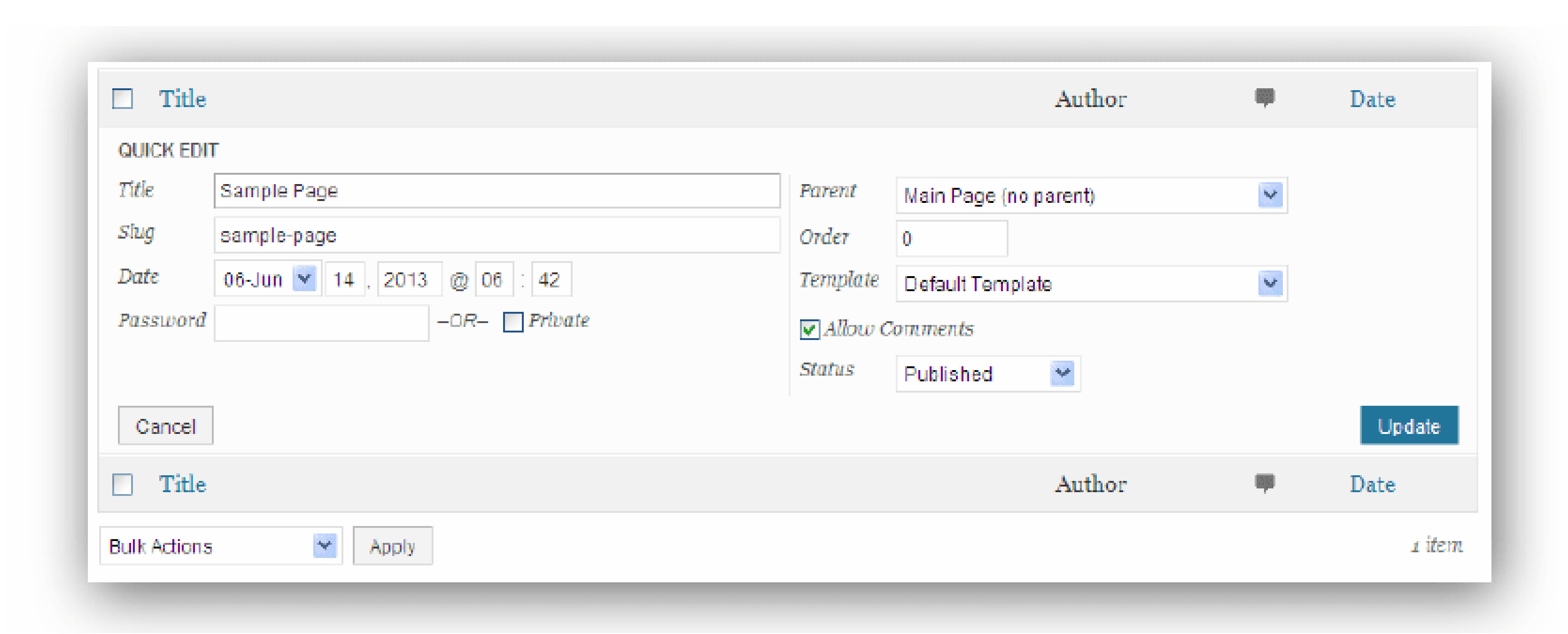


图 14-11　快速编辑页面

我们将表单中的“Title”文本框的内容由“Sample Pages”修改为“Contact Us”，然后单击右下方的“Update”按钮完成修改。这时，我们再返回前台，发现导航菜单中的“Sample Page”已经被替换成了“Contact Us”。

14.1.5　添加博客分类

回到前台后，我们发现之前发布的名为“My First Post”的博客所属的分类为“Uncategorized”。这表明该博客并未归入适当的分类中。在本小节里，我们将为 WordPress 站点添加几个分类，以便更好地管理博客内容。

单击页面右侧的“Site Admin”返回后台，在左侧的菜单列表中，单击“Posts”菜单按钮。然后单击“Categories”链接进入博客分类管理页面，如图 14-12 所示。

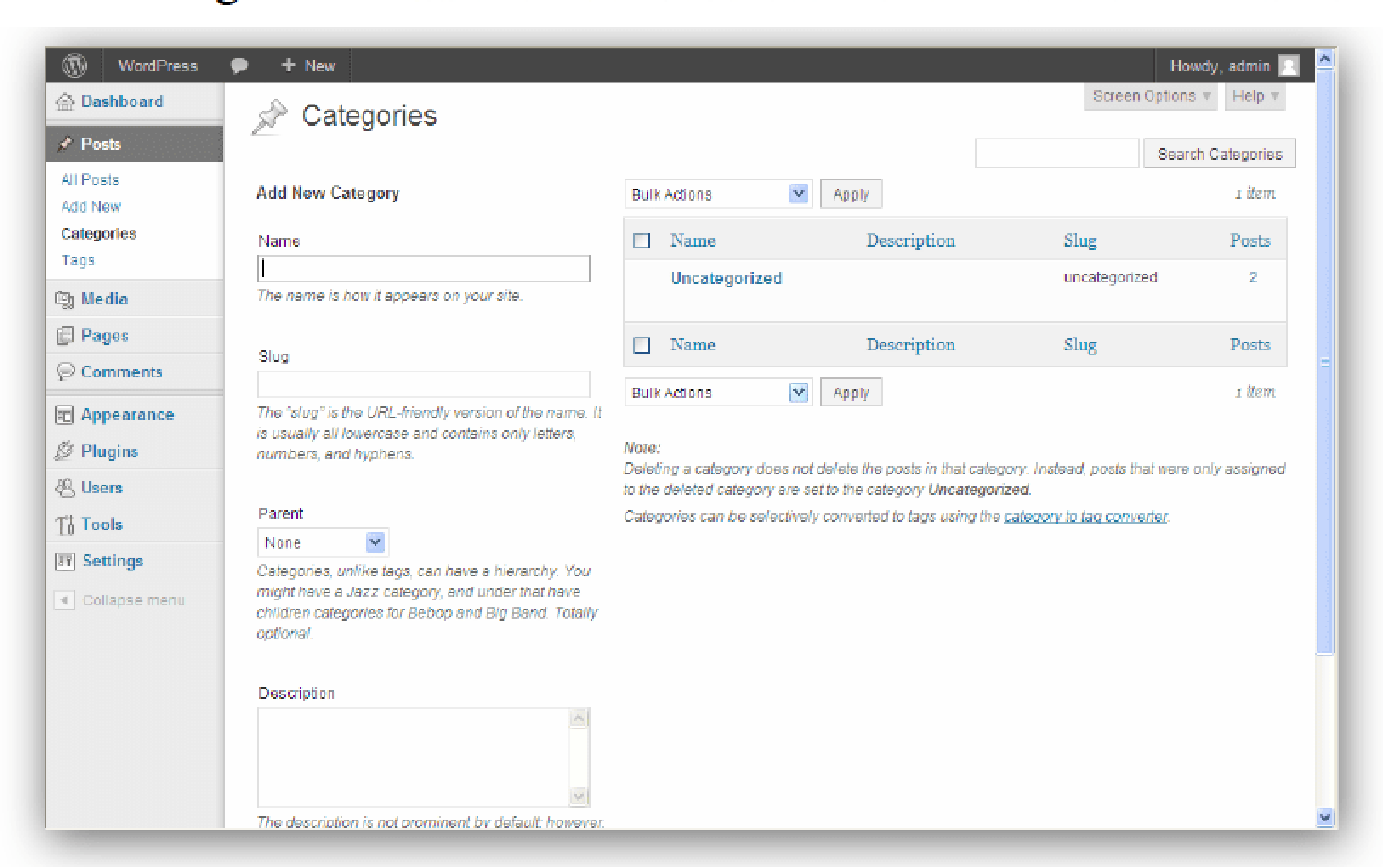


图 14-12　博客类别管理

在这里我们可以添加新的分类，还可以查看、修改和删除已有分类。需要注意的是，预置分类“Uncategorized”可以修改，但不可以删除。如需移除一个自定义的分类，需要

先将其中的博客移至其他的分类中。

在这里，我们打算添加“Writing Project”、“Graphic Design”和“Life Drops”三个分类，然后将“Uncategorized”修改为“Warm Greetings”。在添加和修改博客分类时，需要注意一个名为“slug”的参数的设置，该参数是为了帮助浏览器更好地识别分类的名称。因此不要在 slug 参数中使用空格和其他特殊字符。我的建议是，使用与分类名称相同的内容。如果分类名称中有空格，用“-”替代空格。如果分类名称中有大写字母，在 slug 参数中使用相应的小写字母，如图 14-13 所示。

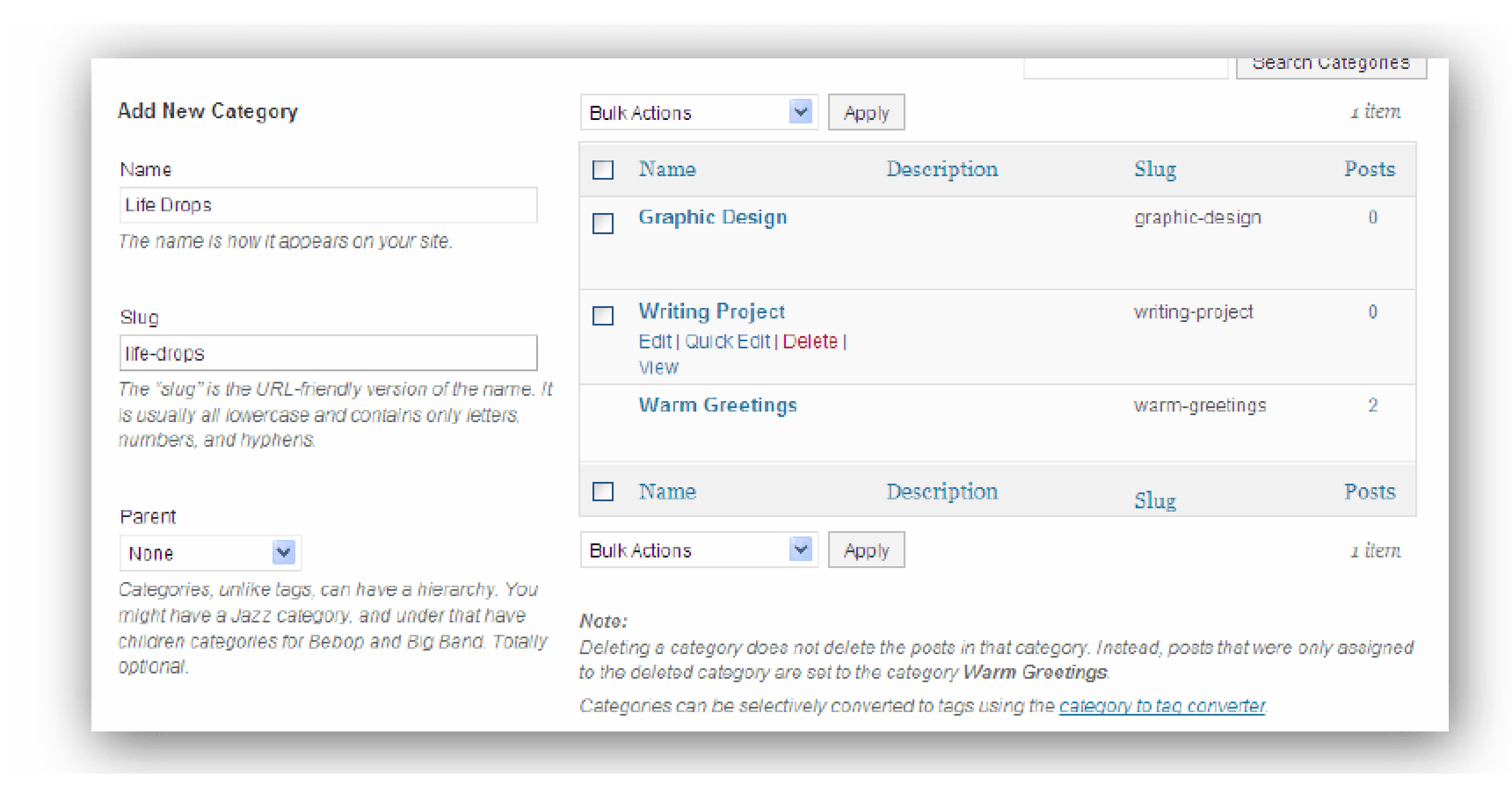


图 14-13　添加新分类

至此，我们完成了博客分类的添加和修改。

14.1.6　管理导航菜单

在添加了博客、页面和分类之后，为了让访问博客的人可以更方便地找到想要的内容，我们打算把各分类和页面做为菜单项添加到导航菜单中。

单击页面左侧的“Appearance”菜单按钮，然后单击其下的“Menus”链接进入菜单管理页面，如图 14-14 所示。

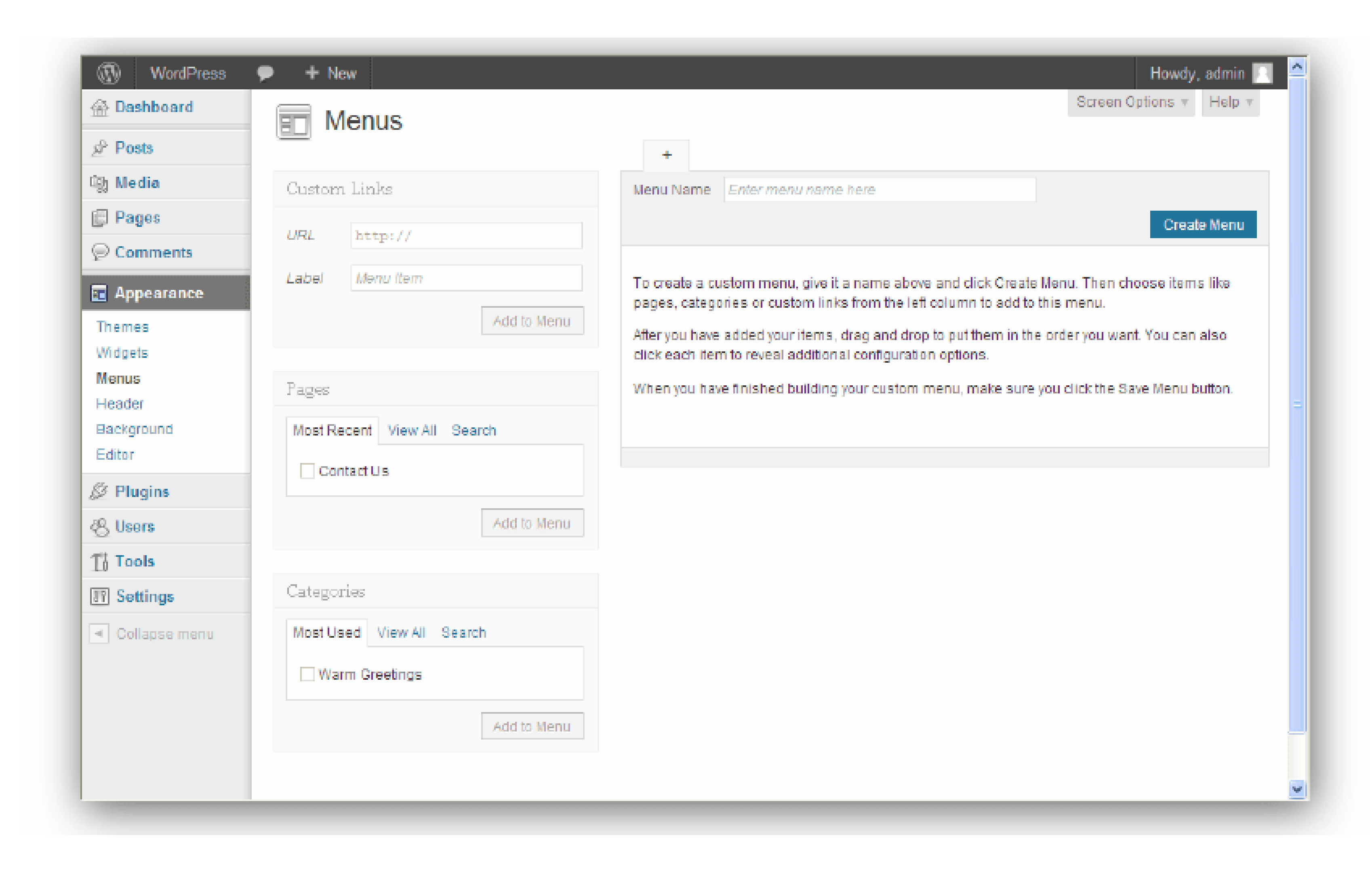


图 14-14　菜单管理

页面的左侧为可向菜单中添加的元素，它们一共有三种：自定义链接、定制页面和博客分类。这意味着我们可以在导航菜单中添加自定义链接、定制好的页面和博客分类。页面的右侧为菜单编辑区域。在编辑菜单之前，我们需要创建一个新的菜单，可以创建的菜单数量与当前使用的主题有关。WordPress 当前版本的默认主题只支持一个菜单。

在填写了菜单名称之后，单击“Create Menu”按钮创建菜单，然后将之前创建好的页面和博客分类添加到菜单中，并通过拖曳为它们排序。当菜单元素排列妥当之后，单击右下角的“Save Menu”按钮保存菜单。随后页面上方出现黄底通知，告诉我们保存菜单成功了。

现在就让我们返回 WordPress 前台查看一下导航菜单的效果吧。首页上赫然出现了刚才添加的元素，单击各个菜单按钮，可以进入相应的博客分类和自定义页面，如图 14-15 所示。很炫吧！

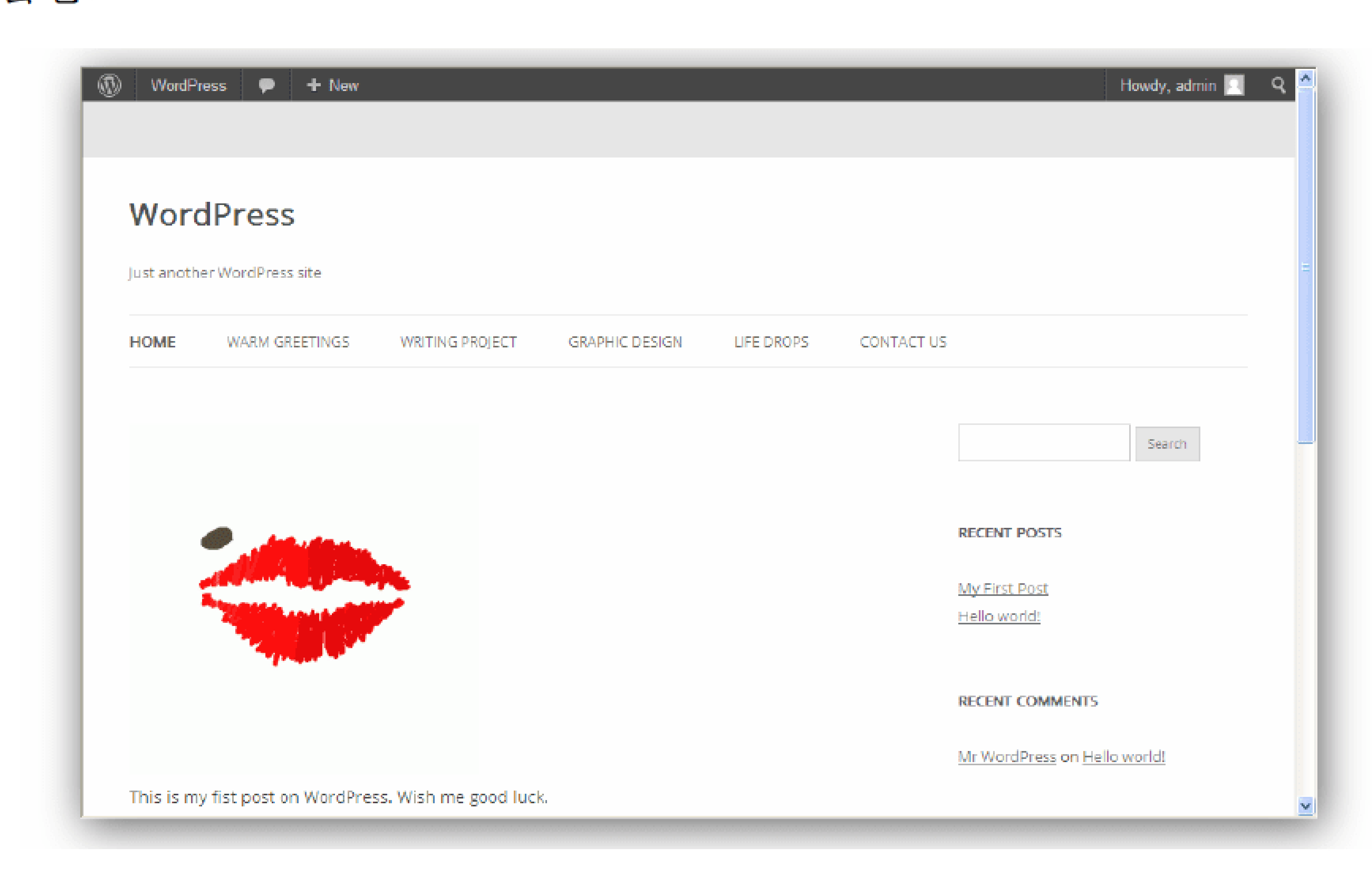


图 14-15　添加了导航菜单的首页

14.1.7　管理前台主题

WordPress 默认的主题，比较适合二次开发。如果不想亲手制作 WordPress 主题的话，也会有很多的选择。单击页面右侧的“Site Admin”按钮回到系统后台，再单击左侧的“Appearance”菜单按钮，进入主题管理页面，如图 14-16 所示。

在主题管理页面上，有两个页签。一个为“Manage Themes”，用于管理主题；另一个为“Install Themes”，用于安装新主题。我们进入第二个页签。然后单击页签下方的一排链接中的“Newest”查看最新发布的主题。在每个主题的截图下方会出现三个按钮，分别是“Install Now”、“Preview”和“Details”。读者可以使用这些链接来安装、预览或者详细查看对应的主题。当选中一个主题后，安装它，然后再返回“Manage Themes”页签激活它。再返回前台看看效果吧，图 14-17 所示就是我应用了新主题之后的效果，是不是大不一样了呢？

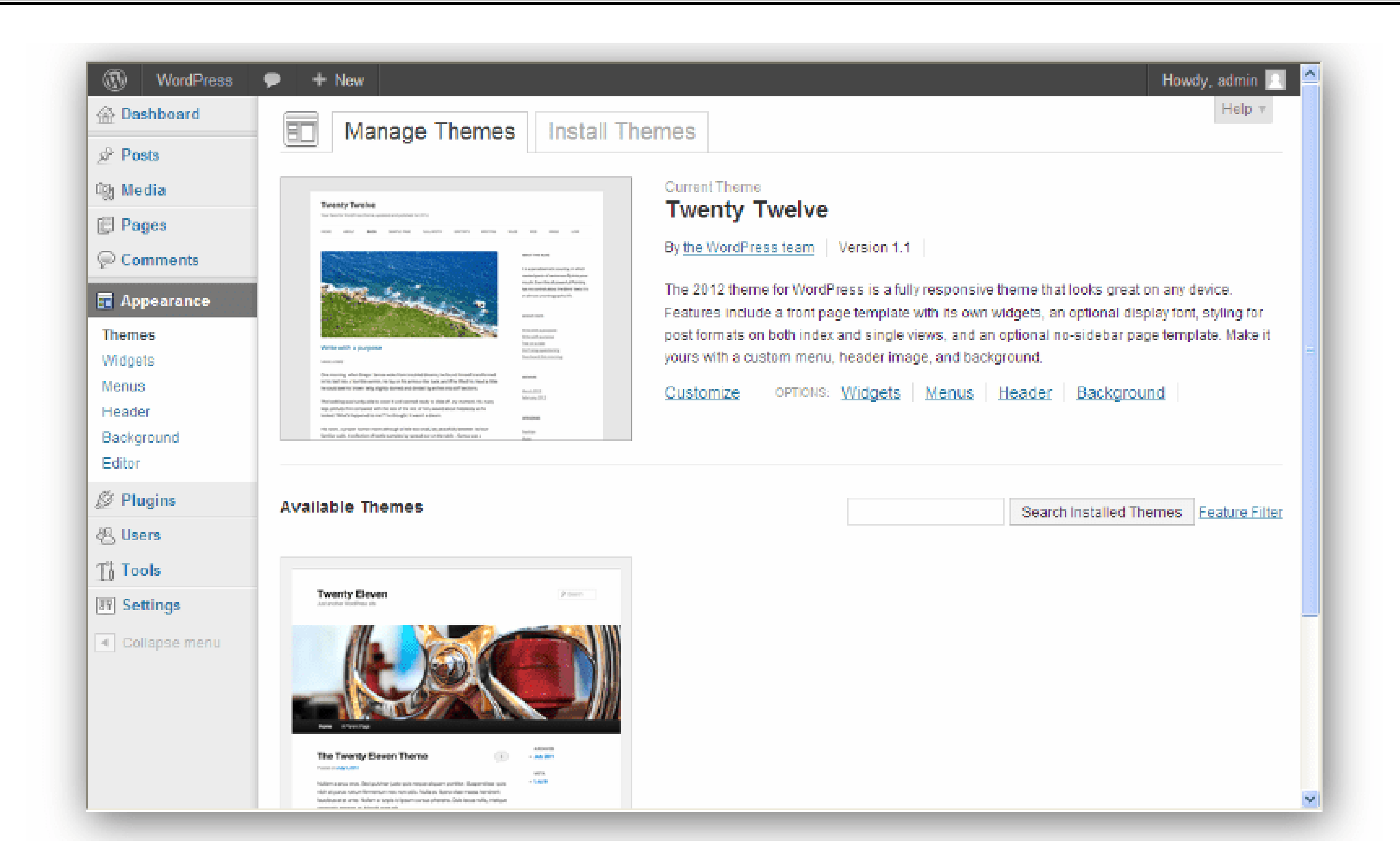


图 14-16　主题管理

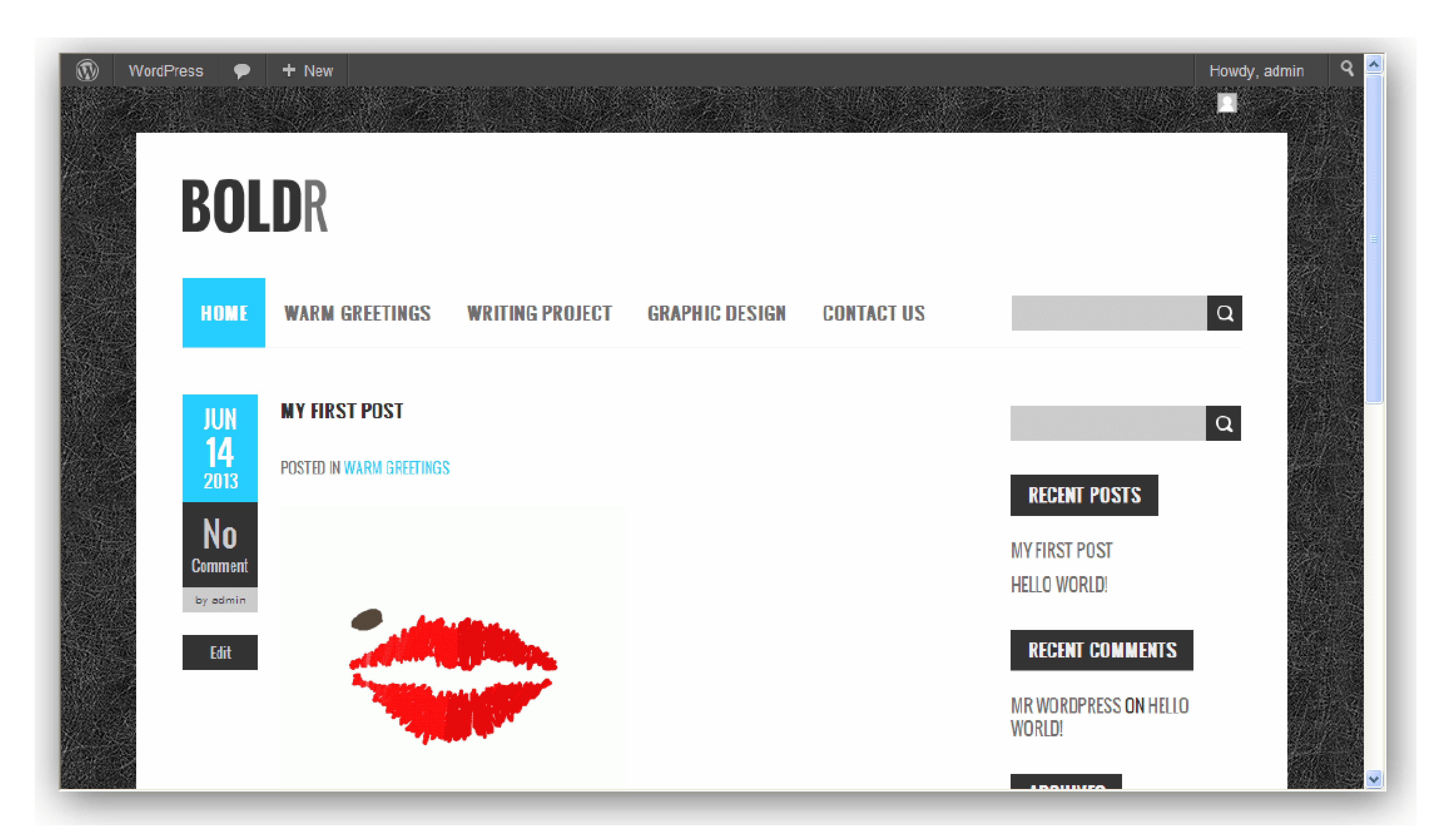


图 14-17　应用主题后的效果

14.1.8　小结

WordPress 做为当今世界上最流行的博客软件，其稳定性和丰富的扩展性是毋庸置疑的。如果读者想要搭建一个个人博客，分享你的所见、所闻及所想，WordPress 绝对是不二之选。

同时 WordPress 也支持二次开发，也可以结合本书中学到的 PHP 知识和 WordPress 网站上的资料，发挥多看、多问和多想的“三多”精神，为 WordPress 开发新的主题和插件。如果有可能的话，我也打算总结一下自己在 WordPress 主题和插件开发方面的经验与大家共享。

14.2　Drupal

提起 Drupal，大名如雷贯耳。它是使用 PHP 语言编写的开源内容管理框架（CMF），它由内容管理系统（CMS）和 PHP 开发框架（Framework）共同构成。连续多年荣获全球最佳 CMS 大奖，是基于 PHP 语言最著名的 Web 应用程序。

Drupal 是一套开源系统，全球数以万计的 Web 开发专家都在为 Drupal 技术社区贡献代码。因此，Drupal 的代码在安全性和健壮性上具有世界最高水平。这也是美国白宫、美国商务部、法国政府、纽约时报、SONY 等著名政府和机构纷纷采用 Drupal 建设网站的最重要的原因。

但是，与 WordPress 等使用模板建站不同，Drupal 的学习曲线相当漫长和陡峭，比 PHP 难度大得多，要求也高得多。事实上，只有精通 XHTML、CSS、Javascript、PHP 和 MySQL 的开发人员，经过长期刻苦的学习，才有可能真正的驾驭 Drupal。

在本节里，我们将简单地介绍一下 Drupal 的安装和使用。

14.2.1　安装 Drupal

在 Drupal 的官方网站上，我们可以看到关于 Drupal 安装的介绍。如果你和我一样是在本地测试 Drupal 应用程序，那么可以将下载到的 Drupal 包文件（.zip）解压到本地 Web 服务器根文件夹下。

随后打开浏览器，在地址栏中输入 http://localhost。如果在浏览器中看到了类似于如图 14-18 所示的内容，表明 WordPress 可以在本地服务器上安装了。

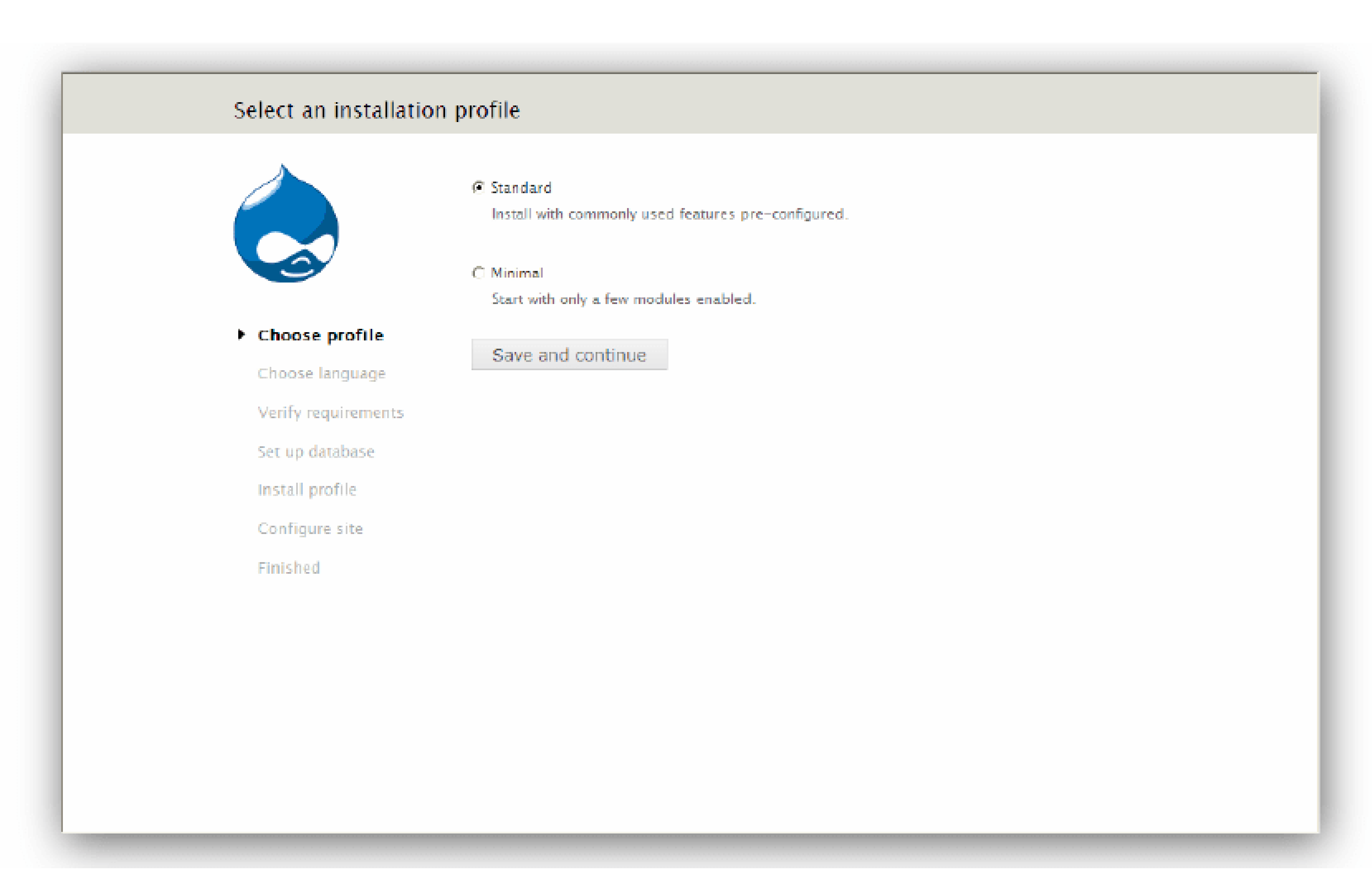


图 14-18　Drupal 安装起始页面

它的安装过程与 WordPress 类似，都需要在本地先为其创建一个 MySQL 数据库。因此，我们再次打开 PHPMyAdmin，创建一个名为 Drupal 的数据库，然后关闭 PHPMyAdmin。现在我们回到如图 14-18 所示的页面，选择“Standard”标准安装模式，单击“Save and

Continue”按钮保存当前设置并继续。

按照页面提示，下载中文语言包，并将其放在指定位置，然后重新加载当前页面使设置生效。当出现如图 14-19 所示的页面时，选择“Chinese, Simplified（简体中文）”，然后单击“Save and Continue”按钮保存设置并继续。

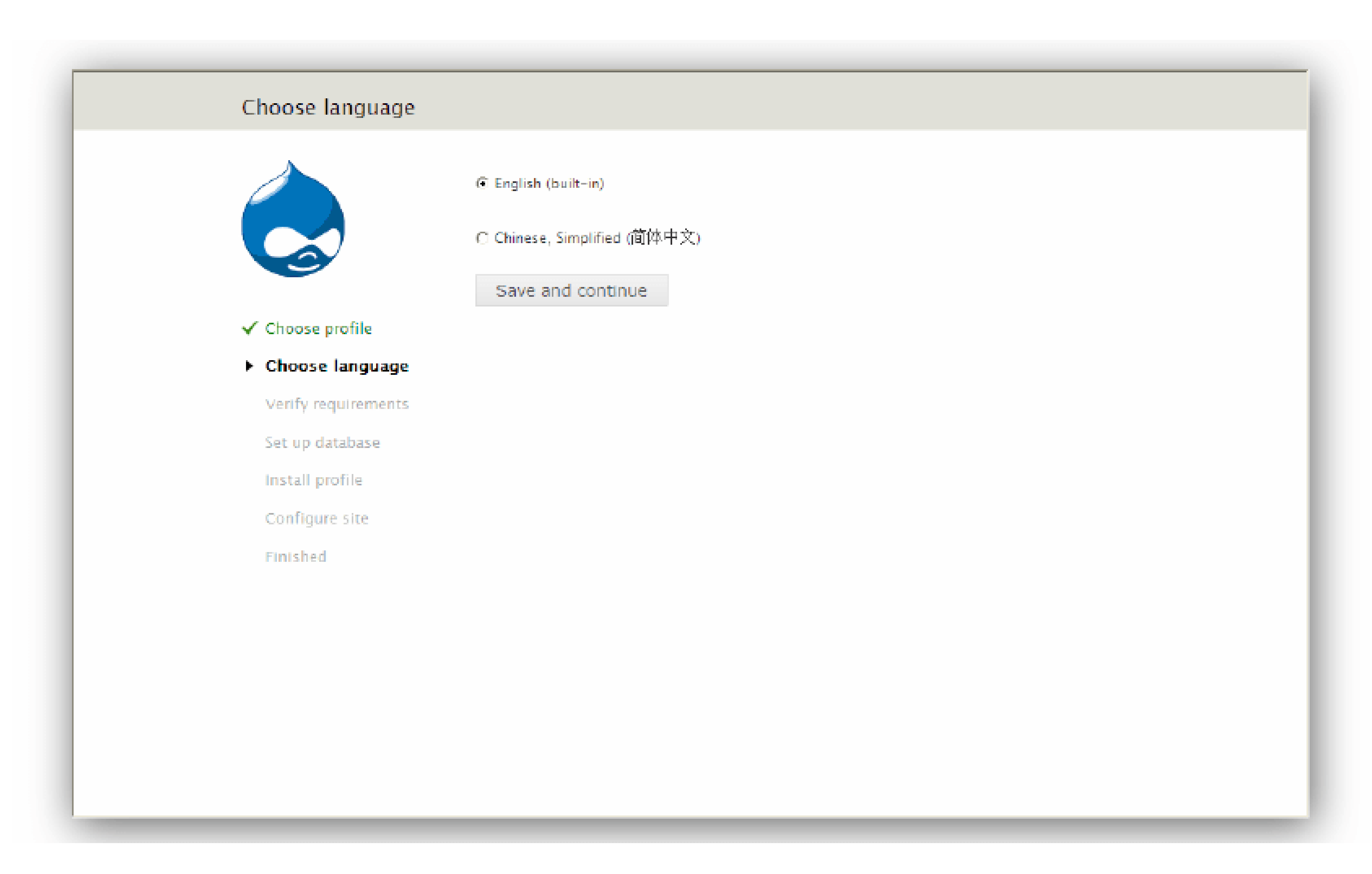


图 14-19　选择 Drupal 界面语言

这时，Drupal 的安装界面就自动变成简体中文了。现在我们填入“数据库名称”、“数据库用户名”和“数据库密码”，然后单击“保存并继续”按钮保存当前设置并继续，如图 14-20 所示。需要注意的是，在本地测试 Drupal 程序使用默认的高级设置即可。若在远程服务器上测试，且 MySQL 数据库与 PHP 不在同一台服务器上，则有可能需要对“高级选项”里的“数据库主机”和“端口”进行设置。



图 14-20　填写数据库信息

这时，Drupal 就开始安装了。安装程序会先按设置完成安装，然后安装本地化文件。接着，我们需要手动设置一下网站，这一点与 WordPress 相同。在这里需要设置网站名称、网站邮箱、管理员账户、服务器所在地和当地时区。当完成这些设置后，单击“保存并继

续”按钮，系统将保存当前设置并完成本地化文件的安装。

当我们看到如图 14-21 所示的页面时，表明我们的安装成功了。这时，我们可以单击页面上的“访问新网站”链接来访问安装好的 Drupal 应用程序。

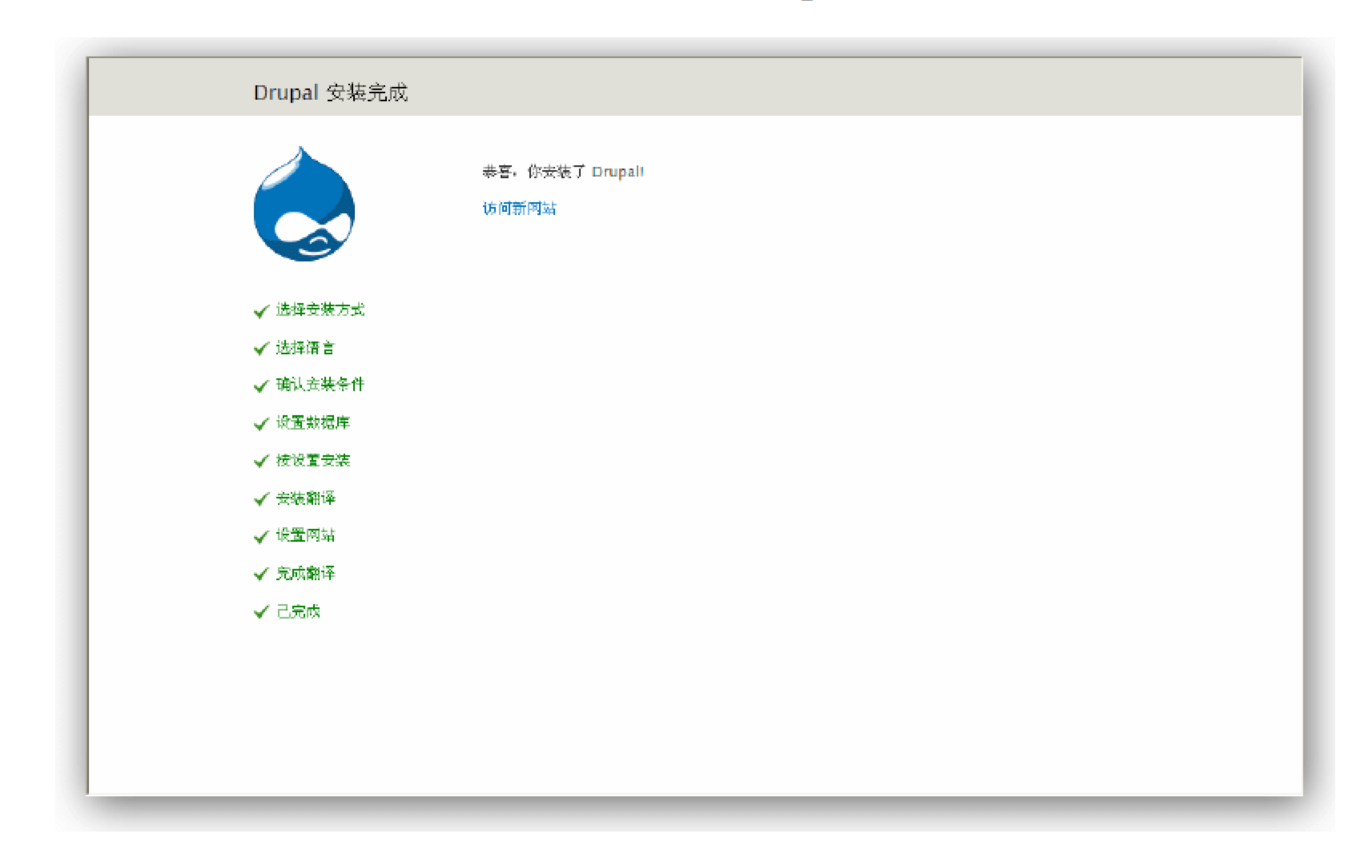


图 14-21　Drupal 安装成功

14.2.2　了解 Drupal 的使用方法

在打开网站首页后，系统会自动以你在安装过程中创建的管理员账户登录。如果没有，你可以手动输入用户名和密码登录 Drupal。当成功登录 Drupal 后，你会发现首页顶端出现了一条黑底的工具条，如图 14-22 所示。这也是 Drupal 与 WordPress 不同的地方。WordPress 为我们提供了一个独立的后台，如果需要在完成一项设置之后查看效果，则需要手动切换到前台查看。而在 Drupal 里，没有所谓前台和后台的概念。如果你是以管理员账户登录的，则可以在页面的顶端看到相应的管理导航菜单。单击管理导航菜单中的项目，则会在当前页面上方出现了一悬浮的区域供你进行管理操作。所有的操作都会实时更新到 Drupal 站点中，可以实时看到设置的效果。



图 14-22　Drupal 的管理导航菜单

在这里，我们可以选择“报告”|“状态报告”命令来查看当前站点的状态。图 14-23 展示了安装在本地计算机上的 Drupal 站点的状态报告。

在这份报告中，可以看到 Drupal 核心的版本、服务器上安装的 PHP 引擎的版本、数据库系统的类型和版本、文件系统是否可写等与管理 Drupal 站点相关的内容，并可以据此

来了解 Drupal 站点的运行情况，如图 14-23 所示。

单击悬浮区域右上角的“⊗”按钮就可以关闭悬浮区域，回到打开悬浮区域之前的页面。



图 14-23　Drupal 站点状态报告

14.2.3　管理站点内容

单击管理导航菜单中的“内容”节点，可以打开“内容”悬浮窗，如图 14-24 所示。



图 14-24　内容管理

在“内容”悬浮窗中，我们可以对站点现有的内容和与之相关的评论进行管理，也可以添加新的内容。在 Drupal 中，默认内容分为“基本页面”和“文章”两种类型。读者也可以在“结构”节点下的“内容类型”中定义新的类型。单击“内容”悬浮窗内的“添加内容”链接，然后单击“文章”创建一篇文章。这时，“创建文章”悬浮窗替代了“内容”悬浮窗，出现在我们面前。

“创建文章”悬浮窗里主要有两个部分。其一为文章内容区域，如图 14-25 所示；其二为文章属性区域，如图 14-26 所示。

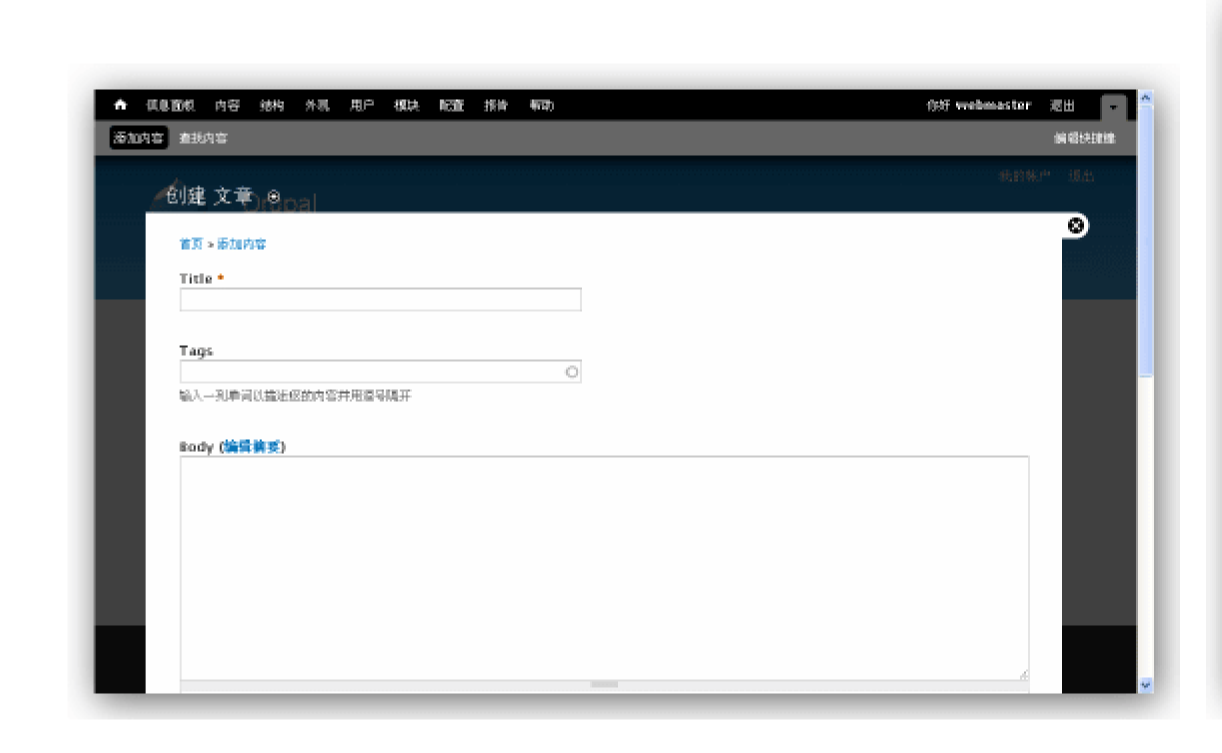


图 14-25 文章内容区域



图 14-26 文章属性区域

在文章的内容区域，我们可以指定文章的标题、标签和内容，这一点与在 WordPress 中添加文章是类似的。如果需要为文章添加摘要，可以单击“Body”旁边的“编辑摘要”链接进行添加。

在文章的属性区域，我们可以发现，Drupal 与 WordPress 相比，有了比较完善的版本管理系统。如果忠实地记录每一次对文章的修订，你就可以很方便地恢复之前的记录。

现在，我们创建一篇文章，标题为《世界，你好！》，然后到 Lorem Ipsum 网站上复制几段话放到“Body”文本框中。单击悬浮窗左下角的“保存”按钮。在悬浮窗消失后，我们就能看到《世界，你好！》这篇文章被放到了首页上，同时在文章上方出现了一个绿框提示，告诉我们《世界，你好！》这篇文章已经成功创建了，如图 14-27 所示。

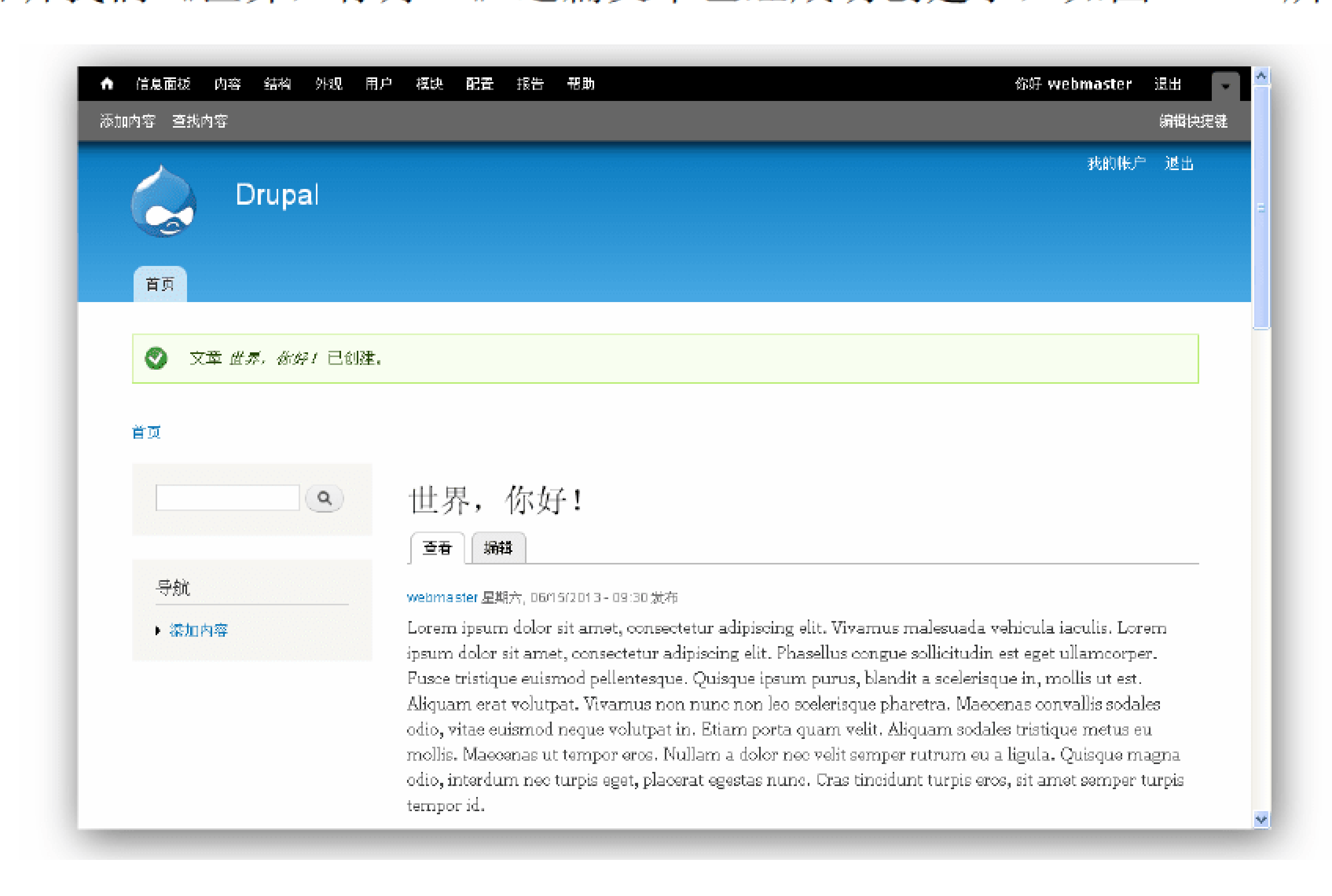


图 14-27 成功创建题为“世界，你好！”的文章

由于，我们现在仍然是以管理员的身份登录 Drupal 站点的，因此，在文章标题下方有两个页签，一个为“查看”，另一个为“编辑”。如果对文章的内容不太满意，可以单击“编辑”页签，在弹出的悬浮窗中对文章的内容进行修改。现在，我们觉得文章的倒数第二段的最后一句不太满意，想把它删除。那么，我们可以单击“编辑”页签，在弹出的悬浮窗中删除倒数第二段的最后一句，然后在文章属性区域的“修订版本信息”中添加对本次修订的描述。

当单击悬浮窗左下角的“保存”按钮后，悬浮窗消失。在《世界，你好！》下方的页签数量由之前的两个，变成了现在的三个。新增加的页签名为“修订版本”，单击它，我们可以查看之前的修订记录。在“修订版本”页签中，可以方便地恢复或者删除相应的记录，如图 14-28 所示。



图 14-28　《世界，你好！》的修改版本记录

添加“基本页面”与添加“文章”类似。我们可以在“基本页面”中创建一个名为“关于自己”的页面，将其 URL 别名指定为“aboutme”。具体的操作方法这里就不再赘述。

14.2.4　管理站点结构

大家或许已经发现了，在 Drupal 的“文章”编辑区域并没有像 WordPress 一样为我们提供一个可以方便设置分类的选项。因为，Drupal 对站点的管理方式更加灵活。而通常情况下，灵活就意味着复杂。这里我们首先需要了解一下 Drupal 的站点结构。

Drupal 站点上的所有内容都是以“节点”的形式存储在数据库中的。也就是说，我们发布的每一篇文章，编辑的每一个页面、乃至于投票、论坛帖子等等都是一个个的节点。这一点大家可以从上一小节里发布的文章的 URL 中看到。《世界，你好！》这篇文章的 URL 是“http://localhost/node/1”，也就是说《世界，你好！》这篇文章是当前 Drupal 站点中的第一个节点。以后我们创建的各种文章也好、投票也好、论坛帖子也好，怎么给它们分类呢？

现在，我们单击管理菜单上的“结构”，打开“结构”悬浮窗，如图 14-29 所示。

在“结构”悬浮窗中，我们可以对 Drupal 站点的“内容类型”、“分类”、“区块”和“菜单”进行管理。



图 14-29　Drupal 站点的“结构”悬浮窗

为了给发布的文章分类，我们得先建立一个分类。在“结构”悬浮窗上单击“分类”按钮，然后在出现的“分类”悬浮窗里单击“添加词汇表”链接为 Drupal 站点添加一个词汇表。词汇表名称确定为“类别”、描述为空。创建成功后，系统自动返回“分类”悬浮窗。这时窗口中的列表里出现了刚才添加的名为“类别”的词汇表，同时，窗口的上部也出现了一条绿框黄底的提示信息，告诉我们词汇表创建成功了，如图 14-30 所示。



图 14-30　添加词汇表

现在，单击“类别”行右侧的“添加术语”在类别下新建一些类别。下面是我打算添加的一些文章的分类以及它们的权重和 URL 别名，如表 14-1 所示。所谓“权重”指的其实是它们排列的先后顺序；而 URL 别名，则可以让我们很方便地通过 URL 找到这些分类。

表 14-1　当前Drupal站点所需文章分类

权　　重	分 类 名 称	URL 别名
0	平面设计	graphdesign
1	前端开发	frontend
2	生活点滴	lifedrop

添加完这些分类后，回到“类别”悬浮窗，我们就可以看到之前为“类别”词汇表添加的几个分类了，如图 14-31 所示。



图 14-31　为“类别”词汇表添加术语

由于我们为每个术语都设定了一个 URL 别名，因此，可以通过 URL 别名来访问它们。以“平面设计”为例，我们可以在浏览器的地址栏中输入“http://localhost/graphdesign”来访问。同理，也可以分别使用“http://localhost/frontend”和“http://localhost/lifedrop”来访问“前端开发”和“生活点滴”两个分类。

好了，现在我们需要修改一下“文章”类型的表单结构，以便在添加新文章的时候可以为其分类。单击管理导航菜单上的“结构”，然后单击“结构”悬浮窗里的“内容类型”列出当前系统中所有的内容类型。找到“文章”一行，单击其右侧的“管理字段”链接进入“文章”悬浮窗的“管理字段”页签。

在该页签里的“添加新字段”下方的文本框里输入“articlegroup”，然后将“字段类型”设置为“术语来源”、控制设置为“选择列表”。接着单击下方的“保存”按钮保存当前设置，如图 14-32 所示。

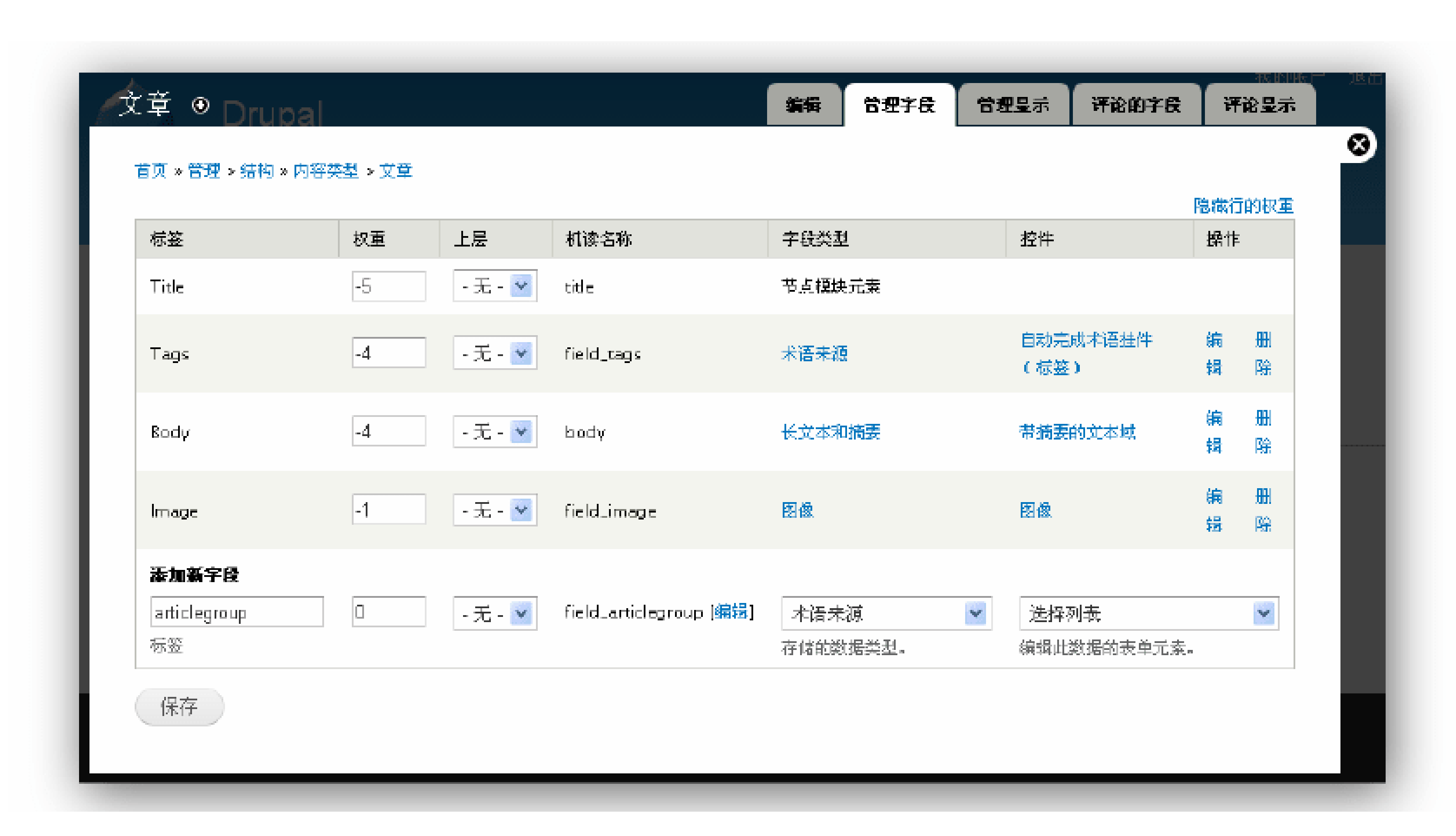


图 14-32　为“文章”类型的内容添加新字段

然后在弹出的“articlegroup”悬浮窗中为该字段指定名为“类别”的词汇表。这也是我们之前创建的那个词汇表，单击“保存字段设置”按钮。这时，系统会返回“Article Group”悬浮窗，可以进一步设置这个字段。例如，我们可以设置该字段为必填项，默认值为“生活点滴”等。

至此，文章分类我们就处理好了。

现在，我们想把之前创建的那篇文章放到“生活点滴”栏目下，应该如何操作呢？

单击管理导航菜单上的“内容”，在“内容”悬浮窗下方的列表里找到《世界，你好！》

这篇文章。单击其右侧的“编辑链接”按钮进入文章编辑页面。拖动页面找到“Article Group”字段，选择下拉列表中的“生活点滴”，然后单击“保存”按钮保存当前修改。

接着我们在浏览器的地址栏中输入“http://localhost/lifedrop”来访问生活点滴栏目。会发现《世界，你好！》这篇文章已经安静地出现在“生活点滴”这个标题下。

可是有一个问题，这样访问各个分类会不会麻烦了一些。是否可以把这些分类添加到主菜单中呢？答案自然是肯定的。

单击管理导航菜单上的“结构”按钮。在“结构”悬浮窗中单击“菜单”进入“菜单”悬浮窗，悬浮窗中列出了当前系统所有的菜单。找到“主菜单”，单击其右侧的“列出链接”链接进入该菜单的链接列表。我们发现当前“主菜单”里只有一个名为“主页”的链接。单击列表上方的“添加新链接”，然后将上面的三个分类添加进主菜单中。其中，“菜单链接标题”为各分类的名称、“路径”为各分类的 URL 别名、权重与各分类在“类别”词汇表中的权重相同。添加完成后，回到“主菜单”悬浮窗的“列出链接”页签，查看已经添加的各菜单链接，如图 14-33 所示。



图 14-33　成功添加分类至主菜单中

当检查没有问题之后，关闭“主菜单”悬浮窗回到 Drupal 站点首页。我们发现之前添加的各菜单链接都出现在了主菜单中，如图 14-34 所示。单击“生活点滴”按钮，也可以查看到《世界，你好！》这篇文章。

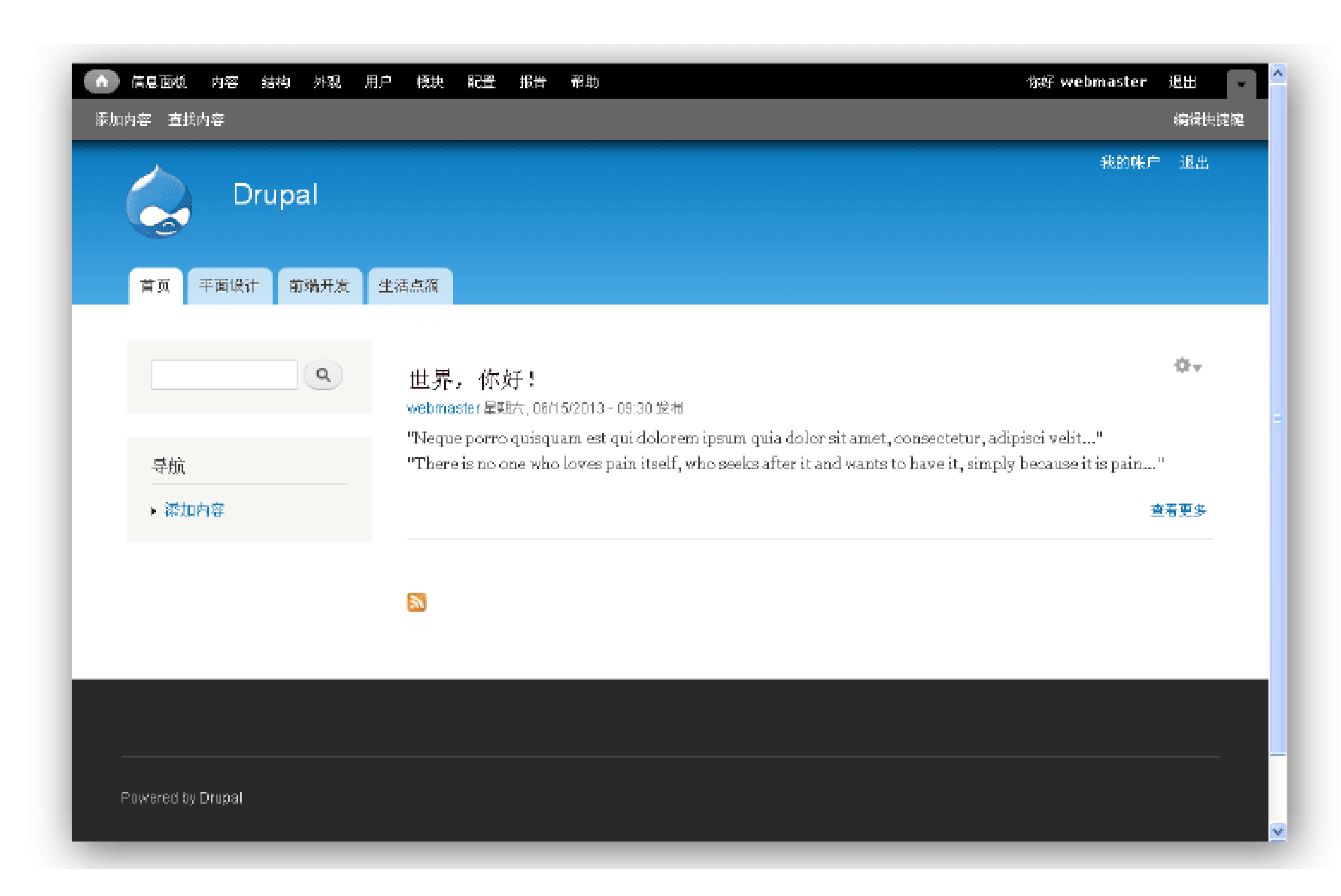


图 14-34　已添加的分类显示在主菜单中

接下来，我们可以把上一节里添加的那个“基本页面”也做为主菜单中的一员添加进来。具体怎么做，大家自己想想吧。完成以后的效果如图 14-35 所示。

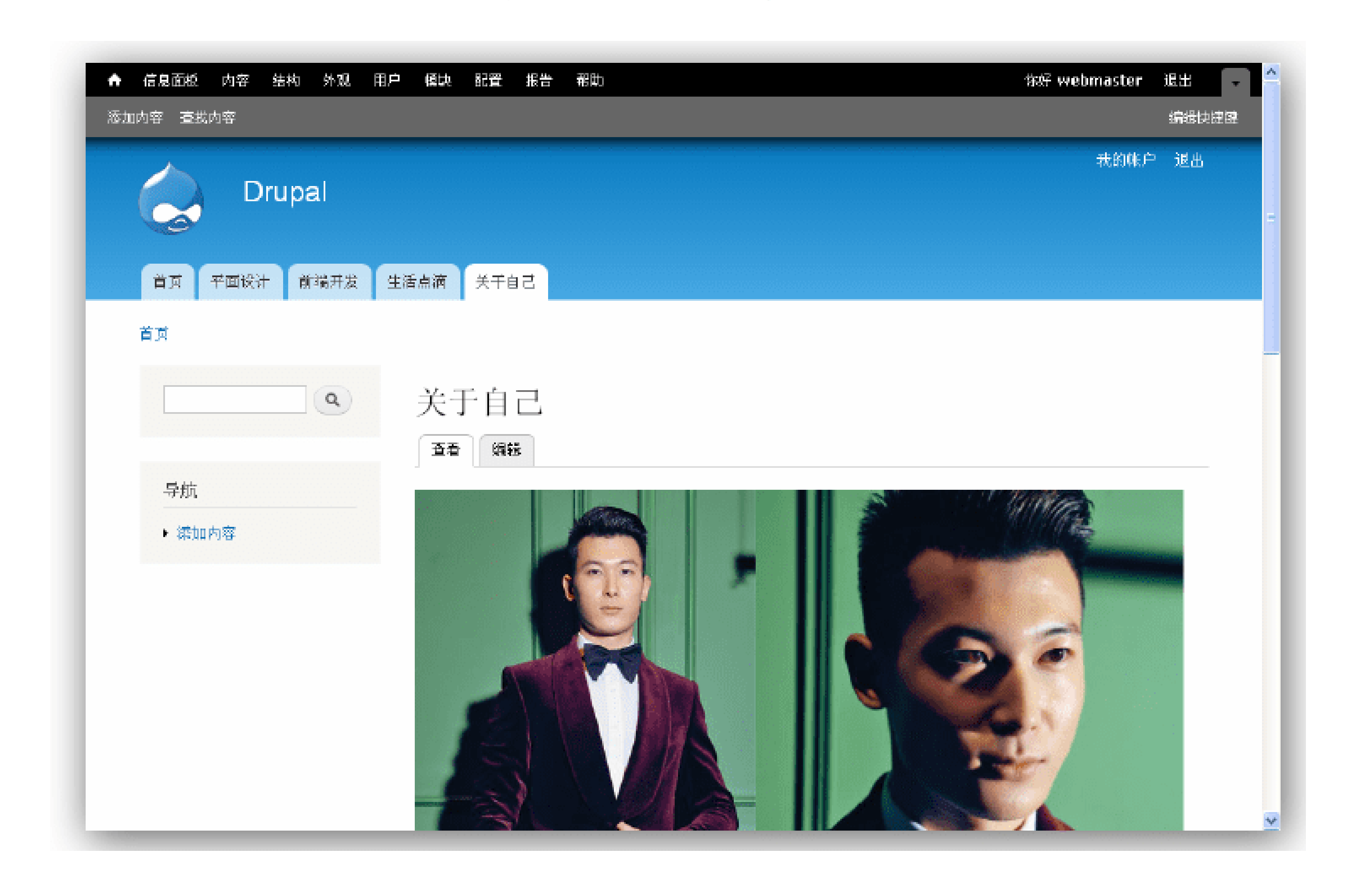


图 14-35　添加“基本页面”到主菜单

14.2.5　管理用户

用户管理对于一个专业的网站来说是十分必要的。不同类型的用户都会接触到同一个系统不同的方面。为了便于管理、同时也为了网站和服务器的安全，对不同类型的用户给予合适的授权是一个不错的主意。

Drupal 对用户权限的管理是通过“角色”来实现的。所谓“角色”，我们可以把它看做是拥有相同权限的一群用户的集合。通过“角色”，我们可以对网站实现分级管理，同时也可以根据访问者属性的差异，为其提供合适其阅读的内容。

单击管理导航菜单中的“用户”链接进入“用户”悬浮窗。在这个悬浮窗中一共有两个页签，一个为“列表”，另一个为“权限”。在“列表”页签中，我们可以查看当前系统中所有的用户，并可根据不同的条件过滤出需要查看的用户，如图 14-36 所示。



图 14-36　Drupal 的用户管理——列表

在“权限”页签中，我们可以为不同角色的用户分配不同的权限。同时也可以添加新的角色，如图 14-37 所示。



图 14-37　Drupal 的用户管理——权限与角色

另外，我们还可以通过管理导航菜单上的“配置”项下的“用户”框中的设置来决定谁可以在本站注册，在注册时是否需要提供邮箱地址等等与用户注册有关的内容。我们还可以设置一个 IP 地址的黑名单，由这个黑名单中列出的 IP 地址发起的访问都会被拒绝，从而保证服务器的安全。可以说 Drupal 的用户管理功能灵活性很大，但是如果了解了它的工作原理，使用起来还是十分方便的。

14.2.6　小结

在这一小节里，我们了解了 Drupal 做为一个内容管理框架的各项基本功能和使用方法。你也可以使用本书中学到的 PHP 知识，结合 Drupal 网站上的学习资料，使用 Drupal 架设一个功能完善、灵活性高和安全性强的网站。

Drupal 和 WordPress 一样，也有着丰富的扩展和主题，如果你想在 Drupal 的基础上进行二次开发也会非常容易。